NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF A THREE-TIERED WEB-BASED INVENTORY ORDERING AND TRACKING SYSTEM PROTOTYPE USING CORBA AND JAVA

by

Ahmed Otoom

March 2000

Thesis Advisors: Daniel R. Dolk
James Bret Michael

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2000	Master's Thesis

TITLE AND SUBTITLE : Design and Implementation of a Three-tiered Web-based Inventory Ordering and Tracking System Prototype Using CORBA and Java

5. FUNDING NUMBERS

6. AUTHOR(S)
Ahmed Otoom

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Many enterprises are still running and maintaining several operating system and platform dependent legacy applications. The variety of platforms and operating systems poses a challenge to system-wide interoperability and performance, increases the cost of maintenance, locks enterprises into certain vendors, and leads to a lack of an adequate information infrastructure which results in a waste of computer resources, manpower, and time. In this thesis, we have designed and implemented a component-based three-tiered Web-based Inventory Ordering and Tracking System (IOTS) prototype that demonstrates the technical feasibility of making an enterprise's applications both interoperable and scalable on a system composed of multiple platforms and different operating systems. The prototype uses CORBA, an industry-backed, non-proprietary, standard-based distributed architecture and Java, a high-level object-oriented language that enables enterprises to leverage the use of the Internet and benefit from the enhancements in the client/server and the decrease in the prices of desktop computers. The prototype demonstrates how to overcome the problem of the stateless nature of HTTP and build the Object Web where Java applets run on the IIOP. The prototype's source code can be tailored to some specific business requirements and enterprises having problems similar to those addressed may benefit from this research and adopt its development methodology.

14. SUBJECT TERMS
Interoperability, Reengineering, Inventory Ordering and Tracking, CORBA, Java, Database, electronic commerce, Internet, and Web-Database Connectivity

15. NUMBER OF PAGES
228

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	19. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

DESIGN AND IMPLEMENTATION OF A THREE-TIERED WEB-BASED INVENTORY ORDERING AND TRACKING SYSTEM PROTOTYPE USING CORBA AND JAVA

Ahmed Otoom
Captain, Jordanian Air Force
B.S., Mu'tah University, 1992

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT
AND
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

ABSTRACT

Many enterprises are still running and maintaining several operating system and platform dependent legacy applications. The variety of platforms and operating systems poses a challenge to system-wide interoperability and performance, increases the cost of maintenance, locks enterprises into certain vendors, and leads to a lack of an adequate information infrastructure which results in a waste of computer resources, manpower, and time. In this thesis, we have designed and implemented a component-based three-tiered Web-based Inventory Ordering and Tracking System (IOTS) prototype that demonstrates the technical feasibility of making an enterprise's applications both interoperable and scalable on a system composed of multiple platforms and different operating systems. The prototype uses CORBA, an industry-backed, non-proprietary, standard-based distributed architecture and Java, a high-level object-oriented language that enables enterprises to leverage the use of the Internet and benefit from the enhancements in the client/server and the decrease in the prices of desktop computers. The prototype demonstrates how to overcome the problem of the stateless nature of HTTP and build the Object Web where Java applets run on the IIOP. The prototype's source code can be tailored to some specific business requirements and enterprises having problems similar to those addressed may benefit from this research and adopt its development methodology.

DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at risk of the user.

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS/DEDICATIONS

My sincerest appreciation and thanks go to my thesis advisors, Professor Daniel Dolk and Professor James Bret Michael for their sincere patience, support, and guidance.

Also, I would like to thank the Chief of Computer Center/Jordanian Air Force (JAF), COL Saleem Abu Deyah for nominating me to enroll in the Information Technology Management master degree program.

Finally, I would like to dedicate this thesis to my parents (Ali and Moneera), wife Khadijah, brothers (Mohammed, Sultan, Aisa, Sameer, and Samer), brothers-in-law (Nasri and Wasfi), sisters (Mona, Manwa, Ibtisam, Amal, Samar, Sawsan, and Wafa), daughters (Moneera, Mona, and Roqayya), son Abedallah for their sincere love, confidence, and unselfish support, and to my fellow officers at the Computer Center/JAF.

__ Ahmed A. Otoom

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Many enterprises are still running and maintaining several computerized legacy systems that play a significant role in the success of their operations. These legacy systems try to handle every possible requirement a developer could conceive. Such systems are hard to develop, difficult to maintain, and almost impossible to adapt to future needs in a reasonable amount of time. Moreover, the emergence of Web-based technology, enhancements in the client/server architectures, the decrease in the price of personal computers, and the existence of platform-independent programming languages, have opened new ways to do business. These new technologies are considered revolutionary compared to what was available just a decade ago. Therefore, many of these enterprises are considering a redesign process for their legacy applications to benefit from these technological enhancements and to conduct their businesses in a more efficient and economical way.

A single enterprise, depending on the kind of business it provides, may maintain several information systems simultaneously. Those systems may run on different operating systems and on multiple platforms. In most cases, those information systems are not separate from each other. In fact, they are related; that is, the output of one system is often the input used by one or more of the other systems. Enterprises find themselves needing to aggregate or extract information from more than one system. The variety in platforms and operating systems poses a challenge to system-wide interoperability and performance, increases the cost of maintenance, and ties those enterprises to limited choices of technology. For example, every time an enterprise wants to procure or develop a new computerized system, it must know if and how this new system will interoperate with the current legacy systems. Moreover, as a result of being platform dependent, these computerized systems usually suffer a lack of adequate information sharing. This results in redundant data maintained by different systems and leads to a waste of computer resources, manpower, and time. This inadequate information infrastructure also increases the potential occurrence of errors and contributes to a lack of data integrity, for example, an update is missed or is not performed appropriately.

B. OBJECTIVES

One avenue along which to address the challenges mentioned above is to use a platform-independent, object-oriented application language and a backbone that easily connects different applications and databases running on different operating systems. The primary objective of this research is to develop an object-oriented web-based Inventory Ordering and Tracking System (IOTS) prototype as a proof of concept for our proposed first step of implementing interoperability and addressing other shortcomings in the current legacy systems.

Since each enterprise has its own problems, needs, and requirements, there is no way to design an IOTS that satisfies the business needs of every possible enterprise. Therefore, the IOTS prototype is built for a hypothetical group of warehouses that we are going to call the Quick Supply Incorporation (QSI). The QSI is assumed to be running a legacy platform and operating system dependent IOTS that takes the responsibility for receiving, processing, and tracking customers' orders. The QSI is looking forward to solving the problems of its current legacy system and benefiting from the modern technology enhancements.

The open, object-oriented Web-based client/server, IOTS prototype for the QSI is a vehicle for addressing the weaknesses of current legacy systems, the problem of interoperability, and the issue of operating system and platform dependence. The prototype illustrates how effective solutions, made available by the new technology, can be utilized to solve some of the problems of the legacy applications, while demonstrating the technical feasibility of such solutions. Enterprises that want to benefit from the new technology and replace their platform and operating system dependent applications may find this research beneficial. The provided source code for the IOTS prototype can be tailored to some specific business requirements. Enterprises, who have similar problems to those addressed here, may adopt the development methodology used in this research to address their own specific problems.

In this research, we address the following questions:

- What are the requirements of the new IOTS database and application program for the QSI?

- What are the possible benefits and advantages of this program from both the technical and management perspectives?

- What is an appropriate design for the new IOTS database and application program?

- How extensible is the new IOTS in terms of the use of different types of databases and different operating system environments?

C. LIMITATIONS

The IOTS prototype will be used to highlight this research's major objectives and it will not be very detailed. After we design the new structure and define the multiple user interfaces of the proposed IOTS, we pick just one of these interfaces and go through the design and implementation processes. In most cases, we skip the minor details in order to stay focused on our objectives. We will not handle the data migration from the existing legacy system to the new IOTS. The prototype will not track performance measures that show how well the system operates. In order for this prototype to have a practical value and be ready for deployment, it needs to implement the remaining user interfaces, migrate the legacy data, track performance measures, manage security related issues, and pay attention to the business rules that were not taken care of.

D. METHODOLOGY

Our research passes through the following phases:

- Requirement Analysis. We start by analyzing the current situation and the need for such a database application program. Then, we list the requirements of what a modern IOTS should be. The requirements will be derived from the best business practices in place today. It will target solving the existing problems and will gain benefit from the modern technology enhancements.

- Logical Database Design. We develop a data model using the Semantic Object Modeling technique to capture the user requirements, and then we transform the data model into a relational database design.

- Physical Database Design. We use the Structured Query Language (SQL) to build the IOTS's database schema from the design. The database will be loaded with some sample data for testing purposes. The IOTS prototype will use its GUIs to populate some parts of the database with data. The remaining parts will be populated with sample data using SQL commands.

These parts are supposed to be populated by some other user interfaces that we are not going to design.

- Application Design and Implementation. We design and build a three-tiered database client/server application prototype. The first tier, the client, is the user interface that will be running on the Web browsers. The second tier, the server, can serve both HTTP and CORBA. The server will encapsulate the business logic and interact with the client via CORBA/IIOP. CORBA objects will interact with each other using CORBA Object Request Broker (ORB) and they will talk to the third tier using SQL/JDBC. The third tier consists of the DBMS that will be accessed by the CORBA objects. CORBA/Java will be used to encapsulate most of the third-tier functions.

- Analysis and Recommendation. After completing prototype implementation, we discuss the benefits, advantages, and costs of adopting this program over maintaining the status quo from both a technical and management perspective. We also address the issues of interoperability, scalability, and portability. The thesis also recommends future work.

E. ORGANIZATION OF THESIS

This thesis consists of four parts that build on each other: Part I addresses the problem, analyzes the current situation, and lists the requirements for the proposed system. Part II gives the reader background information required to understand the following parts of this research. This part provides an overview of Relational Database Management System (RDBMS) and Structured Query Language (SQL), Java and Java Database Connectivity (JDBC), and Common Object Request Broker Architecture (CORBA). Part III describes the design and the implementation of the IOTS prototype. Part IV analyzes the benefits of adopting the IOTS over maintaining the status quo and recommends future work. The following is a more detailed outline for each chapter:

- Chapter I: Introduction. This chapter introduces the problem to be addressed, gives a justification and purpose of this work, and lists the basic structure of the thesis.

- Chapter II. Inventory Ordering and Tracking System (IOTS). This chapter describes the current system, its problems, and the requirements of a new IOTS.

- Chapter III. Technological Background. This chapter contains an overview of relational databases and SQL, Java and JDBC, and CORBA.

- Chapter IV. IOTS Relational Database Design and Implementation. In this chapter, we build a data model using the Semantic Object Modeling technique. Next, we transform the semantic data model into a relational database design. Finally, we implement the design and generate the schema.

- Chapter V. Design and Implementation of the IOTS Web Based Application Program Prototype using CORBA and JDBC. In this chapter we start by selecting the Object Request Broker (ORB) implementation from the available ones by different vendors. Next, we go through the application design process using the ORB that we selected. After that, we design and implement the prototype. Finally, we list and comment on some snap shots of the prototype.

- Chapter VI. Analysis and Conclusions. This chapter analyzes the benefits and costs of adopting the IOTS application over maintaining the status quo. It will examine to what degree scalability and interoperability are achieved. This chapter will also summarize the lessons learned, give directions on how to enhance the application prototype, and guide the reader to some future related work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. INVENTORY ORDERING AND TRACKING SYSTEM

This chapter analyzes the current situation, addresses the problems, and lists the requirements for a proposed solution. In the later chapters, we design and implement a prototype of the proposed solution as a proof of concept illuminating the solution's technical feasibility.

The Quick Supply Incorporation (QSI) is currently running a legacy IOTS that takes care of receiving, processing, and tracking customers' orders. The IOTS suffers mainly from two major categories of problems. The first category consists of the problems imposed by the external environment in which the IOTS operates. The latter category has to do with the inability of the legacy IOTS to take advantage of more efficient ways of doing business generated by the recent advancements in many new technologies, such as the Web, client/server architectures, and object-oriented platform-independent programming languages. So, a new design for the IOTS needs to address its environmental problems, in addition to satisfying its new requirements of benefiting from the modern technology enhancements. This chapter consists of the following five sections:

- The structure
- The environment
- The current IOTS
- The problems with the current IOTS
- What a modern IOTS should be

A. THE STRUCTURE

Figure 1 shows the highest-level view of the QSI's structure that consists of: (n) number of warehouses, a Central Management Office (CMO), and (n) number of suppliers. The warehouses are spread over different geographical regions and centrally managed by the CMO. The CMO also maintains certain levels of inventory in its main store. This inventory is made available to fulfill expected customers' orders.

The IOTS controls the inventory across these multiple warehouses; each warehouse retains local responsibility for inventory management, order processing, and order tracking. Tracking at this level is managed by Local Store Managers (LSMs).

Another level of management is performed by Global Store Managers (GSMs) in the CMO. The GSMs are granted much more authority than the LSMs. They are able to control the inventory levels no matter in which store the inventory exists. Usually, every GSM specializes in controlling a specific type or types of inventory. In most cases, LSMs must go through GSMs to reach the suppliers since the procurement of most types of inventory is a centralized function controlled by GSMs.

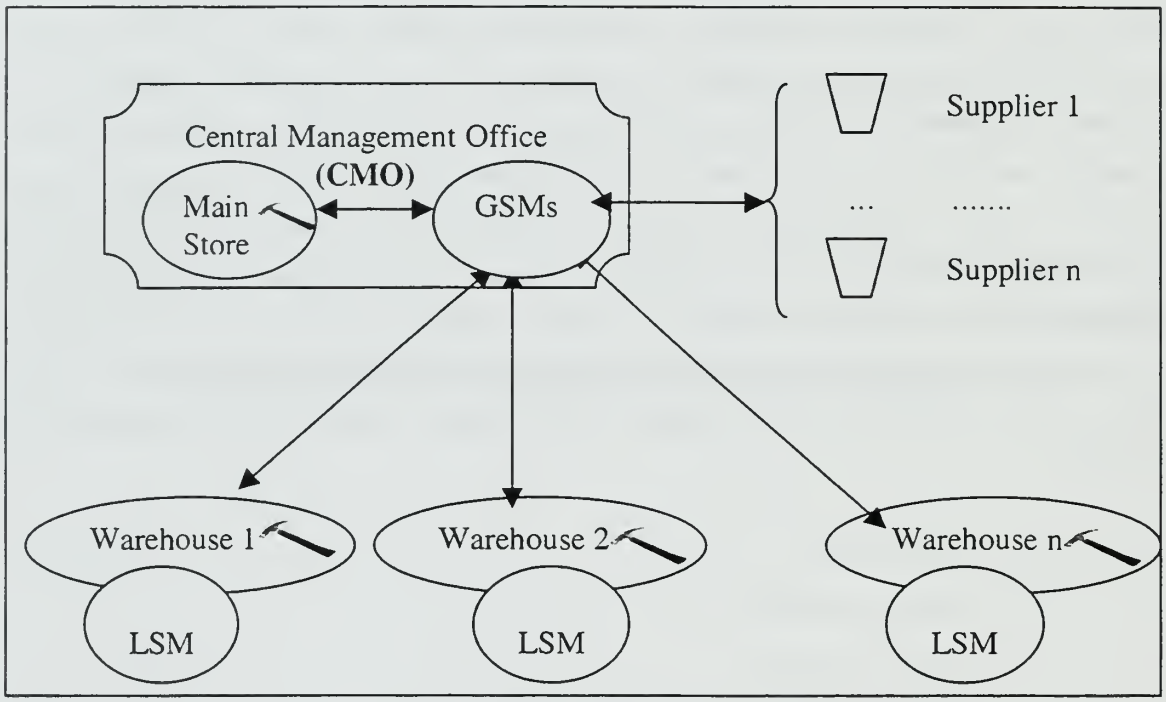


Figure 1. The Structure of the QSI

B. THE ENVIRONMENT

The IOTS is not a closed system operating separately on its own. In fact, it needs to communicate with some other systems to achieve the integrity requirement needed for the QSI to perform its operations successfully. As shown in Figure 2, the IOTS interacts with the Accounting System (ACS), the Personnel System (PERS), and the Management Information System (MIS). Although the interaction among these systems is not as smooth as desired, the QSI views these subsystems as part of a single overall system called the Enterprise Information System (EIS). The EIS is currently running on different

platforms using different operating systems. The Figure shows the environment in which the EIS, including the IOTS, operates:

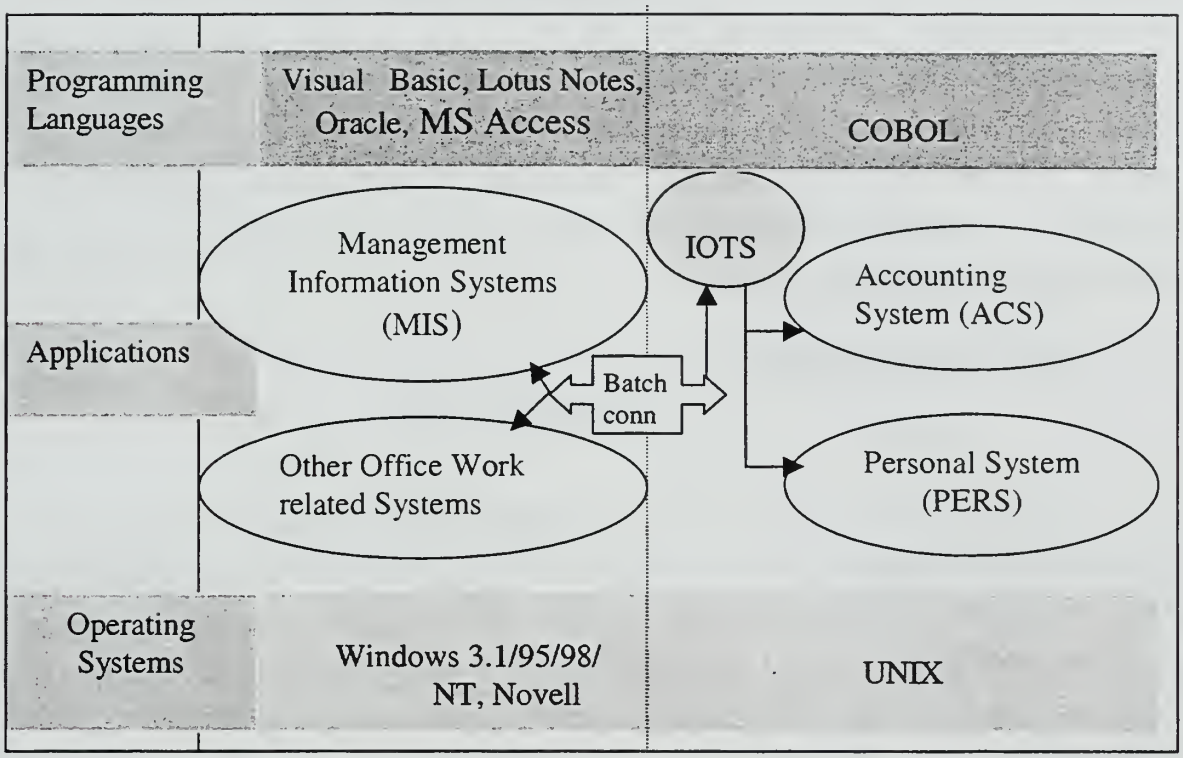


Figure 2. The IOTS Environment

The IOTS interacts with two major categories of applications. These categories are distinguished by the platforms on which they run.

The first category consists of the applications that run on the mainframe, such as the PERS and the ACS. Those applications are mainly written in COBOL, and they run on the top of the UNIX operating system. The interaction between those applications and the IOTS goes smoothly most of the time because of the previously mentioned common similarities among them. To highlight the importance of this interaction, we give two examples. First, users of the ACS need to receive highly accurate status and prices from the IOTS so they can pay the suppliers. Second, the IOTS must verify all of its users with the PERS. In order to be able to track "who did what," the IOTS keeps a history of all the transactions made on any given inventory associated with the identification numbers of the authorized personnel who performed those transactions.

The latter category consists of the applications that run on PC LANs. Those applications are written in various programming languages, such as Visual Basic, Lotus Notes, Microsoft Access, or Oracle, and they run on different operating systems such as Windows 3.1/95/98, or Windows NT. The PC LANs use Novell Netware as their communication backbone. Unlike the natural and smooth real-time interaction among applications in the first category, the interaction between the applications in this category and the IOTS takes place via a batch-processing step. The MIS extracts information from the IOTS in order to enable decision-makers to plan for inventory control, annual budgeting, and cost analyses.

C. THE CURRENT IOTS

1. Analysis

The current IOTS consists of a set of legacy application programs written in the COBOL programming language. The IOTS maintains a centralized database that stores information about all the items available for sale. The IOTS runs an application program that allows the QSI-authorized personnel to access the information in the centralized database, to inquire about inventory levels, to place orders, and to inquire about the status of their orders.

Figure 3 defines the current IOTS system boundary. This figure shows the IOTS network layout, and suggests that the IOTS has the following eight major functional activities:

- **Order entry.** Local warehouses perform this activity. These warehouses are responsible for placing, tracking, and filling customer orders. Another responsibility of these local warehouses is to maintain certain levels of inventory in stock to fill future customers' orders.

- **Shipping.** The responsibility of this activity is to ship ordered items from the main store to the local warehouses. In the current system, customers are responsible for picking up their orders.

- **Receiving.** This activity takes the responsibility of handling the inventory received from suppliers and matching it to the backordered requests.

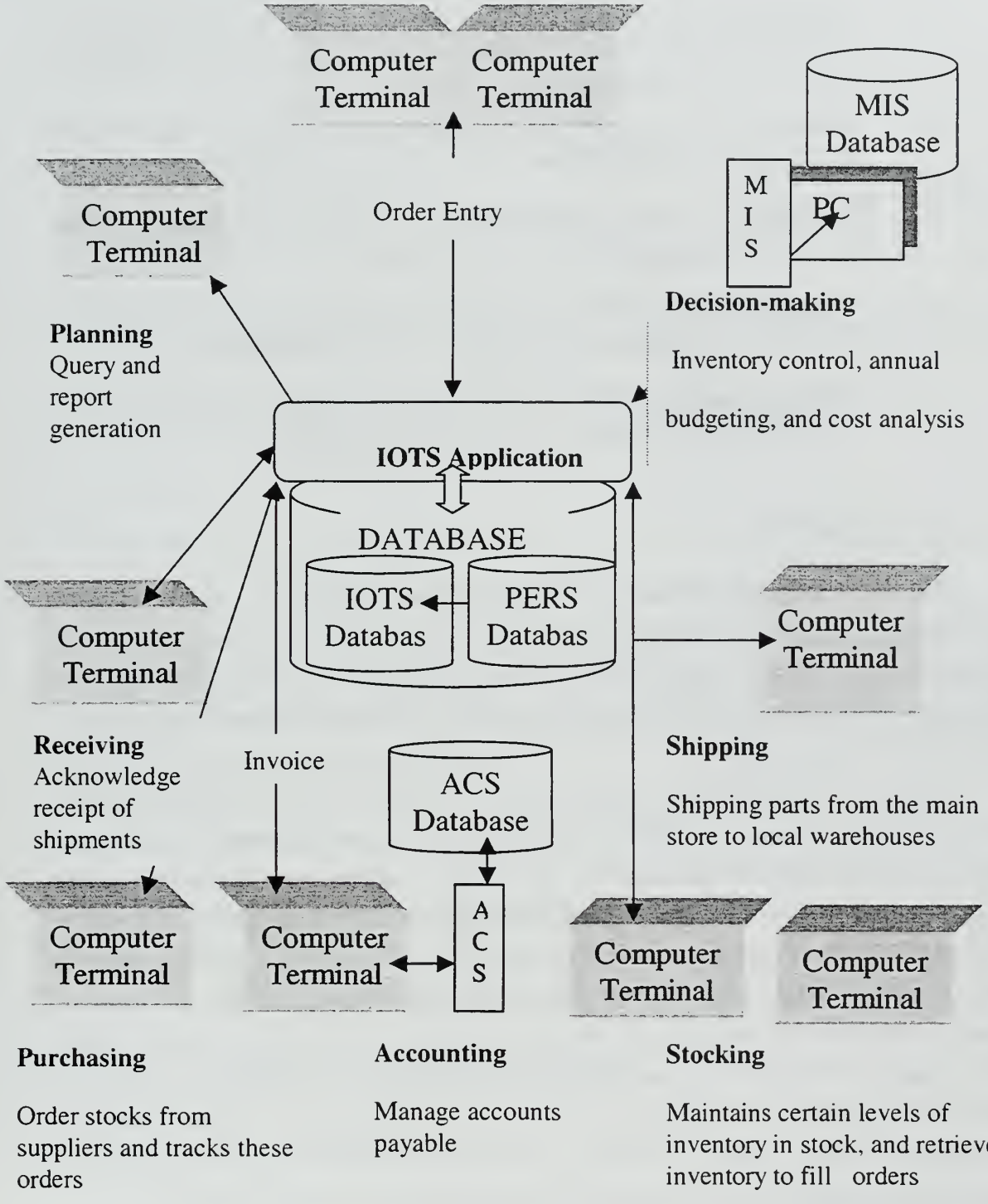


Figure 3. The IOTS System Boundary

- **Accounting.** Once a stock is received from the suppliers, it is processed by the IOTS and the invoice is mailed to the ACS in order to manage the supplier's accounts payable.
- **Purchasing.** This activity is responsible for ordering inventory from suppliers and tracking the status of the suppliers' shipments.
- **Stocking.** This activity is responsible for both placing newly arrived inventory in stock and filling customers' orders.
- **Planning.** This activity uses different statistical reports in order to maintain an appropriate level of stock within budget constraints.
- **Decision-making.** This activity assists top middle managers perform inventory control, annual budgeting, and cost analysis.

2. Data Flow

Data flow diagrams (DFDs) are very useful in representing the overall data flows into, through, and out of the IOTS. DFDs are hierarchical in nature and can be decomposed into smaller, simpler levels. These diagrams are used to facilitate the analysis of the logical data flow in the current system, discover discrepancies, and help us design the new logical system.

The context diagram of the IOTS, shown in Figure 4, presents the highest-level view of the entire system. All DFDs in this research use the DeMarco and Yourdon's symbol conventions that are presented in Appendix B. Basically, the IOTS accepts orders from customers and fills these orders. Once there is a need to get additional inventory, the IOTS places orders to get that required inventory from its suppliers. The suppliers, in turn, ship the ordered inventory with the invoice. The IOTS processes the received inventory, matches the received inventory to the invoice, and sends this invoice to the ACS in order to manage the accounts payable to the supplier. It is important to note that sending the payment to the suppliers occurs between the ACS and the suppliers, and thus is outside the IOTS system. The IOTS makes its inventory levels and some other information available to the MIS that uses this information for its own purposes.

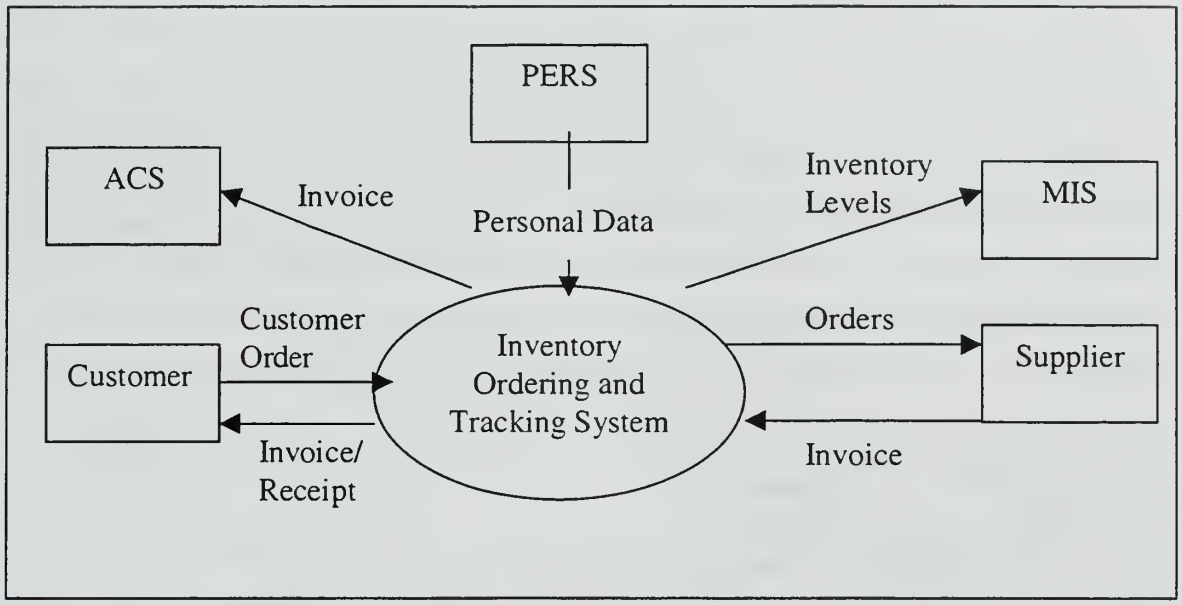


Figure 4. Context Level Diagram for IOTS

In order to get a more detailed understanding of the functionalities of the IOTS, the context diagram can be decomposed further into multiple levels of DFDs. The first level of these DFDs is shown in Figures 5, 6, 7, and 8 by grouping the top level processes into the following four major categories:

- Placing queries
- Processing customers orders locally
- Processing customers orders by Global Store Managers
- Ordering inventory from the suppliers

a. Placing Queries

Placing queries is simply performed by passing requests to the system and receiving responses. The IOTS provides many queries that enable the users to do their jobs without much strain. Appendix C lists some of these queries. Figure 5 identifies the following four major groups of users:

Local Store Managers (LSMs) are able to query the system about the inventory levels of their warehouses, and they can place and track orders. They perform the tracking on behalf of their customers; the customers are informed of their orders' status upon their request. They are also notified when their orders are filled and ready to be picked up. LSMs also track the orders they place for stocking purposes. These orders aim to make some inventory available in stock for future customer needs. Having inventory available in stock will serve the customers without delay, but the inability to determine exactly the future needs of the customers and the budget constraints will make some of those customers experience various delays before getting their orders filled.

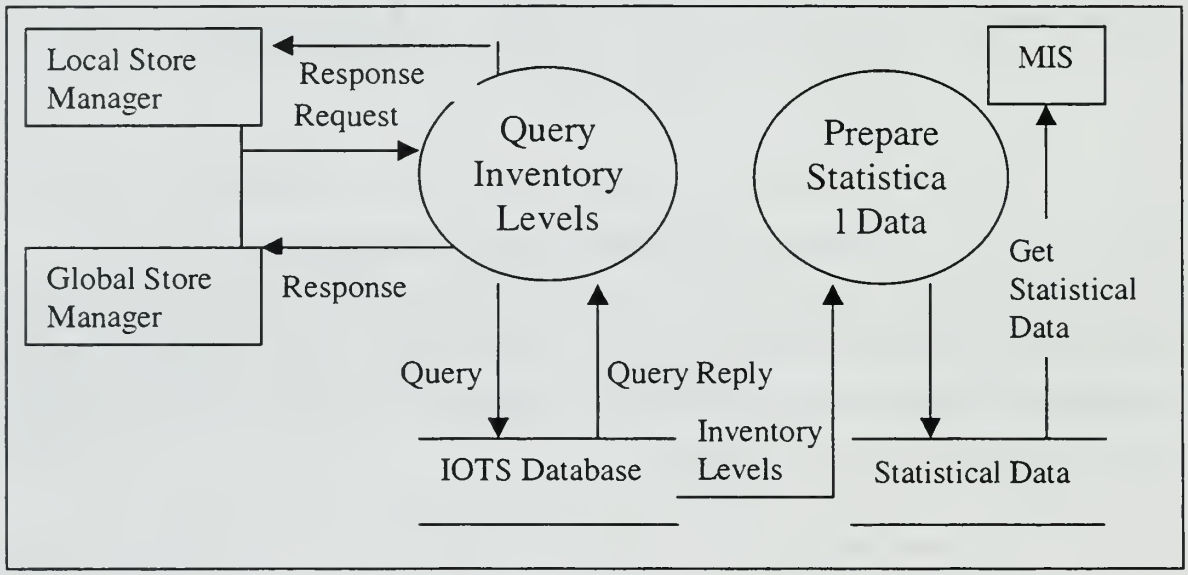


Figure 5. Placing Queries

Global Store Managers (GSMs) are given much more authority than LSMs. The GSMs are able to control the inventory levels no matter in which store this inventory exists. Usually, every manager specializes in controlling a specific type or types of inventory.

MIS users, either strategic or middle managers, receive statistical data from the IOTS for inventory control, annual budgeting, and cost analysis purposes.

Customers perform queries about their orders via the LSMs.

b. Processing Customers' Orders Locally

The LSM places the order on behalf of the customer, verifies that the order has all the necessary information, inquires about the inventory level, and proceeds further in processing the order. The ordered inventory may be available at the local warehouse, partially available, backordered, or not available at all. Figure 6 shows how the customer's order is processed locally:

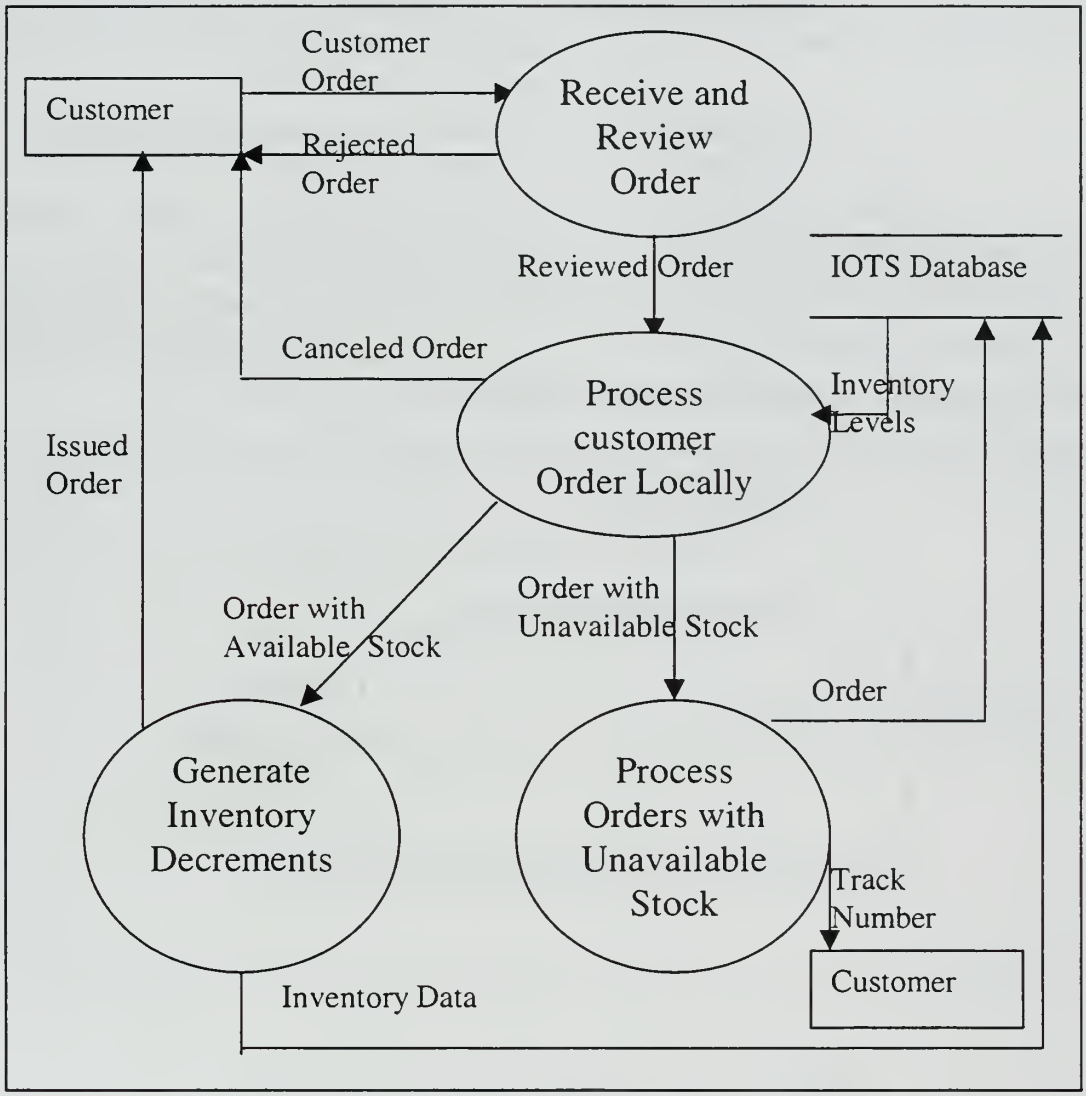


Figure 6. Processing Customer Order Locally

- If the inventory is available the order might be filled in full, or partially filled.

- If the inventory is not available the LSM orders the required inventory from the CMO and the customer is given a tracking number. Once the inventory is available the customer order will be filled. If the customer order was partially filled, the remaining quantity will be filled.

- The LSM may cancel the customer order whether or not there is inventory available in stock.

c. Processing Customers' Orders by the Global Store Managers

GSMs take follow up actions on the orders placed by LSMs. A GSM has control over the inventory available in all the stores. Again, the ordered inventory may be available, in the main store or in any warehouse other than the one that originated that order, partially available, backordered, or not available at all. As illustrated in Figure 7, the GSM processes the orders in the following manner:

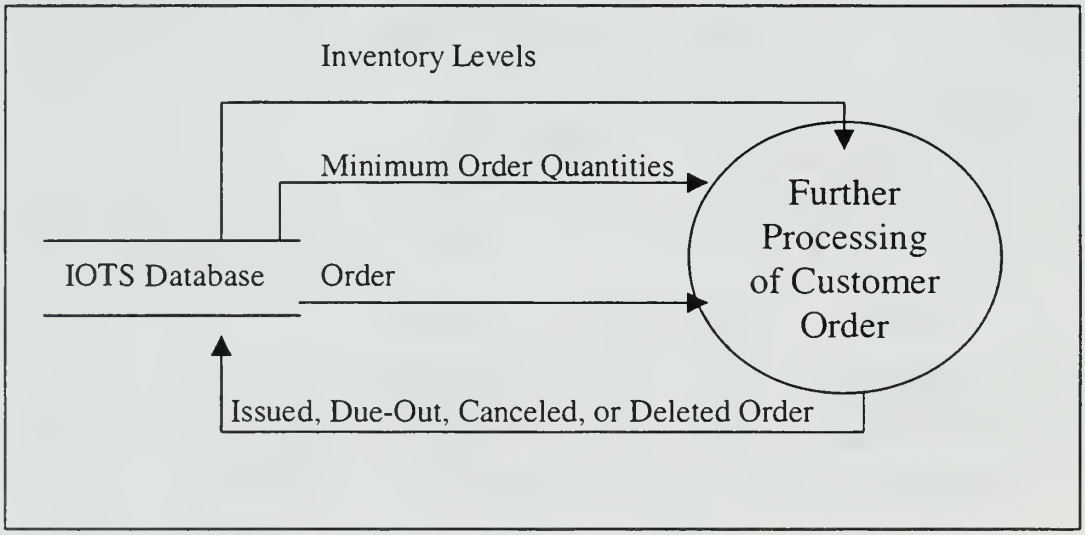


Figure 7. Processing Customer Order by GSMs

- If the inventory is available the order might be filled in full, or partially filled.

- If the inventory is not available the GSM may put the orders in a "Dues out" state in order to get the inventory later from the suppliers. Once the inventory is available, the orders might be filled in full. If the customer order was partially filled, the remaining quantity might be filled.

- The GSM may cancel the orders whether or not there is inventory available in stock.

- Orders that have been filled, but not picked up by the customer for a predetermined period of time, are deleted and become available to be issued to another customer.

d. Ordering Inventory from the Suppliers

This activity is carried out by a specialized staff that reads the inventory levels, the Dues-out orders, and places these orders to the suppliers. The DFD for this activity is shown in Figure 8.

The suppliers will take action on these orders according to their own policies. Once an order is filled and shipped to the warehouse that originated the order, the customer will be notified to pick it up. The processed invoice is sent to the ACS, which will take care of the accounts payable to the suppliers.

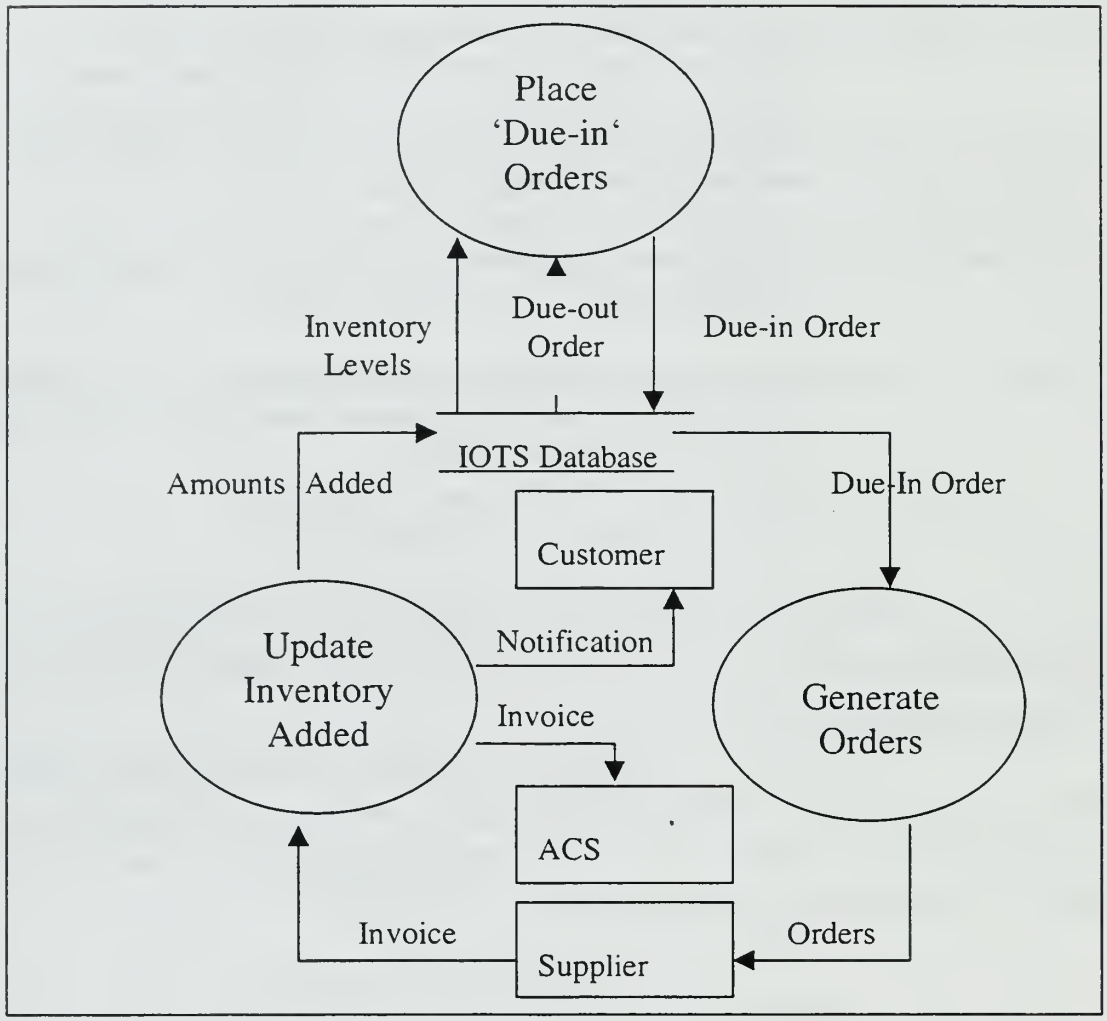


Figure 8. Ordering the Inventory from the Supplier

So far, it is clear that the customer order goes through a sequence of state changes. The availability or the unavailability of the required inventory governs the life cycle of any given order. The state diagram, Figure 9, illustrates how the state of any given order may change and traces the state changes that a customer order may experience during its life cycle. The customer order will be in one, and only one, of the following states:

- **Booked Locally**. The inventory is reserved for the customer and is waiting for the customer to pick it up.

- **Booked by Main**. The inventory is reserved for the local warehouse and is waiting to be picked up.

- **Confirmed/Closed.** The order is filled.

- **Deleted Locally**. The order is deleted due to the fact that the customer no longer needs the inventory already reserved for him or her.

- **Deleted by the GSM**. The local warehouse no longer needs the reserved inventory.

- **Canceled**. The order is either canceled by the LSM or by the GSM.

- **Waiting.** The customer order is waiting to be filled. This waiting state will eventually end in one of the following states: Canceled, Booked Locally, and Deleted by GSM.

- **Due-out**. The GSM places the order in a "Due-out" state, that is, waiting to be ordered from the suppliers.

- **Due-in**. The supplier will fill these orders.

- **Shipped**. The supplier filled the order.

- **Not Available**. The supplier does not, or no longer supplies the ordered inventory.

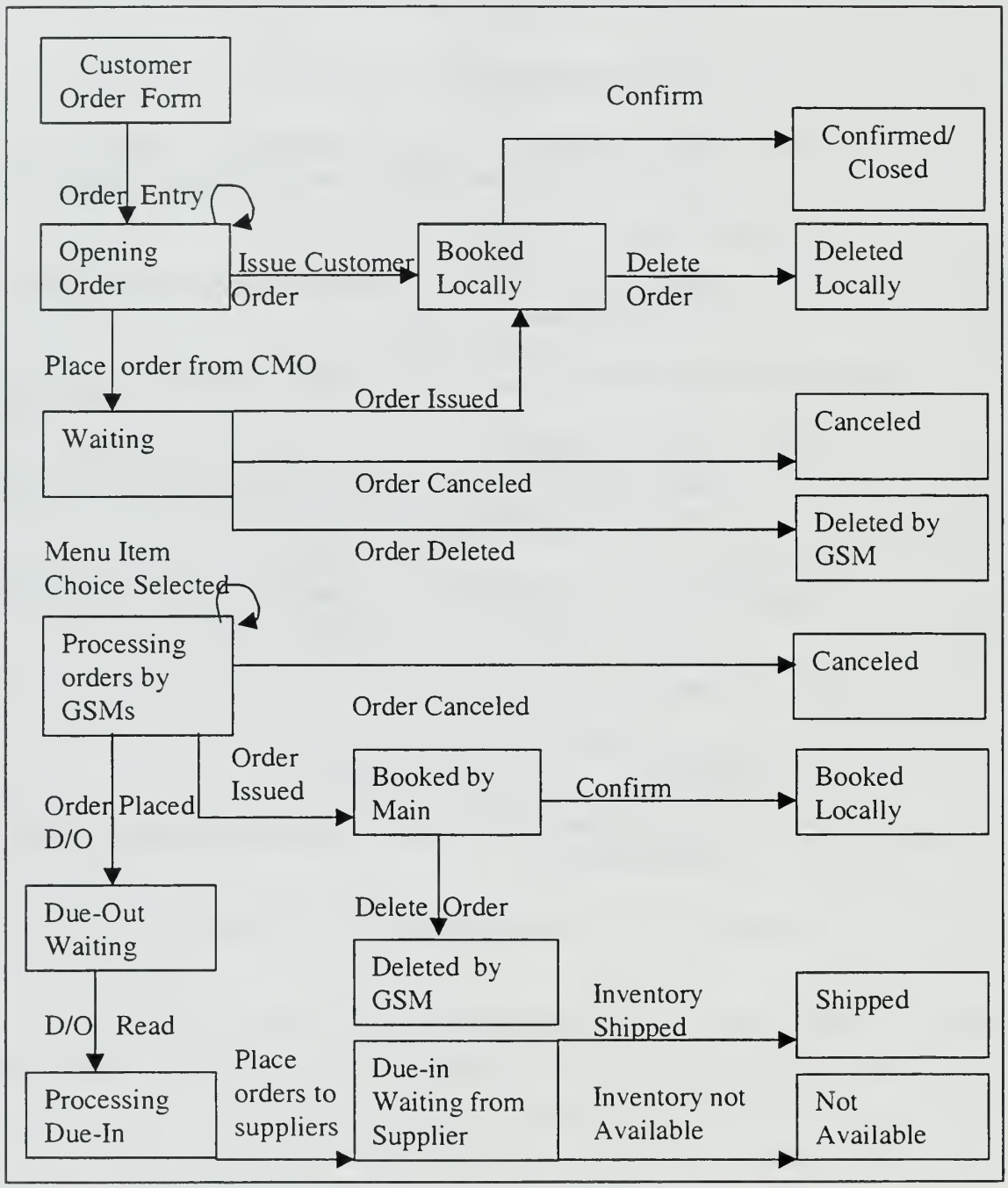


Figure 9. State Transition Diagram

3. How Much Analysis is Needed?

The logical DFDs listed in Figures 5, 6, 7, and 8 can be decomposed further and further in order to obtain more details about the current system. Some experts say that analysts should start as quickly as possible with the new logical DFD, and not spend much time in analyzing the current system (Hoffer, 1996, p. 330). We subscribe to this view for the following reasons:

- The DFDs of the current system tend to give both the analysts and the users more knowledge about the system under analysis. Since both the analysts and the users are familiar with the system, further decomposition of the system will not justify the cost of the time spent on further decomposition.

- The DFDs make it easy to add new functionalities or to eliminate some of the existing ones. The users are mainly satisfied with the current available functionalities, but not with the implementation. Thus, there is a need to focus on the new implementation of the IOTS and not to spend much time in analyzing the functionalities of the legacy applications.

- The major goal of this research is to build an IOTS prototype that addresses the interoperability, portability, platform independence, and how to reimplement the system in a way that gets the benefit of the currently available technology enhancements. Thus, we focus on how the new implementation of the IOTS may address these issues, and what potential benefits the migration to a new open, Web-based IOTS may provide. The QSI needs to form a group of system analysts in order to conduct a more thorough analysis of the current and proposed system. Our experience suggests that a complete study and implementation of such a complex system may take a group, including all of the necessary specialists, from about eight to twelve months.

D. THE PROBLEMS OF THE CURRENT IOTS

The problems listed below were not thought to be problems several years ago. The emergence of the Web-based technology, enhancements in the client/server architectures, the decrease in the price of personal computers, and the existence of platform-independent programming languages introduced new ways of doing business. This new technology is considered revolutionary compared to what was available just a decade ago.

To benefit from these technological advancements and enact business in a more efficient and economical way, we identify the following problems in order to find a solution. These problems fall into two major categories: problems imposed on the IOTS by the environment it operates in, and problems that are specifically related to the way the IOTS was designed and implemented.

1. Problems Imposed by the Environment

The environment affects the IOTS in the following areas:

Portability. The IOTS is designed to work on a specific type of a mainframe and in a UNIX operating system environment. This platform and operating system dependence has substantial impacts on the system. Thus, the QSI is tied to some limited choices of technology; for example, every time the QSI wants to procure or develop a new enhancement on the IOTS computerized system, it must be determined if, and how, this new system will interoperate with the legacy systems.

Cost. The cost of maintaining the mainframe is very high and maintenance requires a high level of expertise that is usually not available in-house and therefore must be outsourced. The maintenance cost of desktop computers however is far less expensive and can be performed in-house. Although putting the power of a large mainframe computer on a desktop is possible, the cost of administering desktop computers that each of which may be configured slightly differently, can be significant and possibly overweigh their advantages. The expected savings may not materialize because of unexpected hidden costs; for example, the savings in desktop hardware acquisition are often offset by high annual operating costs for additional labor required to administer and maintain the network.

The final word on whether the proposed system will provide cost savings or not is left to a specialized detailed cost-benefit analysis which is beyond the scope of this research. The cost-benefit analysis shall consider many issues such as the cost of hardware/software procurement, the cost of maintain the legacy code, the cost of the development of the new system, the cost of administering the system and training the users to adjust to work in the new environment, and the potential benefits of the new system. Therefore, the QSI needs to plan and manage change carefully in order to get the benefit of the new technology.

Interoperability. To circumvent the problem of communicating efficiently between the MIS and the IOTS, a batch process is used. This process extracts data from the mainframe and uses it to update the MIS applications running on the PC LANs. Therefore, the data on the PC LANs are not up-to-date and may not reflect the real situation.

Information infrastructure. The intervening batch process, between the computerized systems on the PC LANs and the ones on the mainframe, gives an example of how vexing the current information infrastructure is. This batch connection process results in redundant data maintained by different systems and leads to a waste of computer resources, manpower, and time. The data redundancy may also lead to a lack of data integrity in case an update is not performed appropriately.

2. Problems in the Design and Implementation of the IOTS

Legacy code. The IOTS application tries to handle every possible requirement the developer could conceive. Such an application is hard to develop, difficult to maintain, and almost impossible to adapt to future needs in a reasonable amount of time.

No GUIs. The users are not comfortable with the text-based user's interface. They need to interact with the IOTS application via a graphical user interface that makes their job easier and more interesting.

Weak supply chain. Customers are not able to place and track their orders, which calls for the QSI to maintain qualified LSMs just for placing and tracking these orders. Moreover, a limited number of customers are reachable. In today's world, not many customers can afford the time to go to a warehouse, place an order, and then come back to pick up their ordered items once filled. Indeed, a reasonable number of customers prefer to get their orders shipped to their places. For the QSI to stay in business, it needs to retain its current customers and look for ways to increase them.

Orders sent to suppliers do not show a real-time state. Suppliers send the states of these orders by fax or mail, and the QSI enters these states to the IOTS in order to refresh the states of these received orders. Therefore, the states of orders on the IOTS database do not reflect the real situation and are not up to date.

The reports listed in Appendix D are generated offline due to the high processing time requirements. These reports are ordered manually from a separate specialized party in the CMO that generates and supplies these reports. Users are satisfied with the

functionalities provided by these reports, but they need to be able to track their ordered reports the same way they track their ordered inventory.

E. WHAT A MODERN IOTS SHOULD BE

In this section we first discuss the business drivers that call for the reengineering of the legacy IOTS. Next, we list the requirements of a new IOTS.

1. Business Drivers

There are two main reasons that call for keeping the current legacy system as it is. First, it provides a vital service that is very risky to disrupt. Second, the current users are used to this system and switching them to a new system requires additional training. Since the reengineering process involves risk and incurs additional cost, there must be strong business drivers that motivate the effort of reengineering before fully launching it. Amjad Umar broadly categories the business drivers that motivate application reengineering as follows (Umar, 1997, p.105):

- Business process reengineering
- New services or business opportunities
- To gain and maintain a competitive edge
- To align IT with business

The reengineering process of the IOTS is motivated or driven by factors that mainly lies in the last three categories listed above. A more detailed study, however, may reveal that this application reengineering process should go side by side with a business process reengineering. The QSI will face a drastic change in the way of doing business and it needs to reorganize in a way that enables it to work in the new open environment. The following are the business drivers that are particularly related to the QSI:

Supply chain management. The supply chain management integrates both buyers and sellers chain processes, which provides the QSI a more competitive advantage and enables it to work in the open environment. As discussed earlier, the QSI is not connected to suppliers in real-time and, therefore, the QSI's records are not up to date. Customers are not able to place and track their orders. The QSI maintains additional staff just to serve customers who call to inquire about their orders. The integration of buyers

and suppliers is "critical for speed and responsiveness in today's hypercompetitive product and service markets" (Nissen). Integrating the supply chain promises the QSI a more efficient way of conducting business. For example, since customers will be able to place and track their orders, the QSI may downsize its customers' service staff, which results in cost reduction, at least in terms of manpower.

Maintain competitive edge and investigate new business opportunities. Many businesses are building new electronic commerce systems in order to gain a competitive edge in this rapidly growing area. The QSI wants to keep pace with this trend and benefit from the new opportunities made available by the emergence of Web-based technology, enhancements in client/server architectures, decrease in the prices of desktop computers, and the existence of platform-independent programming languages.

Legacy code, interoperability, and platform dependence. The QSI wants to overcome the limitations of its legacy code, not be restricted to a specific platform or operating system, and achieve system-wide interoperability.

Cost reduction. Cost reduction is always a goal that drives businesses. The QSI is no exception and needs to conduct its business in a more cost-beneficial way.

2. IOTS Requirements

The requirements of the reengineered IOTS support the business-drivers already discussed and provide the major source of motivation to this research's proposed solution. In the next chapters, we describe our implementation of a prototype system that is based upon these requirements. We explore how implementing these requirements will contribute to the solution of the problems that have been identified. Moreover, the research will investigate what additional benefits will be gained from adopting the modern technology in addition to just solving the existing problems. The IOTS must support the business goals by providing the following requirements:

- Take advantage of the Web technology by directly connecting to the suppliers, enabling the customers to place and track their orders, and having orders shipped to the requested destination.

- Track the status of a customer order across all of the warehouses. Refer to Figure 9, the state transition diagram, and its explanation in the analysis of the current system, for a better understanding of this requirement.

- Consist of software components that can be developed, purchased, rapidly customized, and integrated into heterogeneous multitier platforms.
- Run on all platforms in order to facilitate future integration of IOTS with other systems such as ACS, PERS, and MIS under a common Enterprise Information System (EIS).
- Be easy to maintain.
- Provide cost reduction.
- Provide freedom in selecting application programming languages; applications written in different programming languages must be able to interoperate with each other.
- Provide a graphical user interface that must be:
    - easy to use (i.e., user-friendly)
    - informative and explanatory in terms of providing clear error messages that direct the user to perform a clear action (e.g., mysterious error messages are not acceptable)
    - highly flexible, in order to handle users at all levels of proficiency
    - able to support different access levels to information on a "need-to -know" basis. These levels are:
        - Level zero: This is the highest access degree in which strategic managers will have access to everything in the system. MIS users are the core of this category.
        - Level one. This is the second highest level of access that is associated with the Global Store Managers (GSMs). The GSMs will have access to the entire available inventory, regardless of which store this inventory is located.
        - Level two. This level is associated to Local Store Managers (LSMs). LSMs will have access to the inventory available in their local warehouses only. They are not authorized to query or control items on stores that are not under their span of control.

- Level three. This level is for regular users who will have query privileges only. The access to the IOTS from the ACS will also have this access level.

- Level four. This level is granted to customers accessing the system from the internet. They will be allowed only to place and track their orders.

- Develop a centralized relational database which will store the necessary information about different items offered for sale by the QSI. This database must support Web access.

- Allow the authorized personnel, based on their access levels, to access the information in the centralized database, inquire about the available balance, place orders, and inquire about the status of their orders.

- Provide the queries listed in Appendix C.

- Provide the reports listed in Appendix D. Users must be able to track the status of their ordered reports. Any ordered report will be in one of the following states: Waiting, Canceled, Ready, or Confirmed. "Waiting" means the ordered report is under processing. "Canceled" means the requested report order is canceled and a comment will display the reason of cancellation. "Ready" means the ordered report is ready and can be picked up. The "Confirmed" status indicates that the report was physically handed to the requesting party. The only printing jobs that will be provided online are the ones that do not delay the user for a long period of time, such as printing a limited number of vouchers, invoices, or copies of the screens that are usually generated by the queries listed in Appendix C.

F. SUMMARY

The current IOTS provides the following eight major activities: order entry, shipping, receiving, accounting, purchasing, stocking, planning, and decision-making. The major problem that calls for reengineering the current legacy IOTS is its poor supply chain. In the current system, customers are not able to place and track their orders, the QSI is not connected to the suppliers in real-time, and various inventory ordering and tracking activities cannot easily interoperate with each other. Supply chain management integrates both sellers' and buyers' processes to increase the system's responsiveness and competitiveness by benefiting from the revolutionary advancements in technology. This

Chapter also discusses the business drivers that call for the reengineering process. These business drives provide the major source of motivation to this research's proposed solution. The requirements were listed to support these business drivers and motivate a solution for the identified problems. In Chapter IV we start building the IOTS prototype based on these requirements.

III. TECHNOLOGICAL BACKGROUND

This chapter provides a brief overview of Relational Databases and SQL, Java and JDBC, and CORBA. The purpose of this chapter is to provide the readers who are unfamiliar with these technologies with the background knowledge needed to understand the proposed solution for the IOTS. For more details on the material covered in this chapter, the reader may refer to the list of references.

A. RELATIONAL DATABASES AND SQL

1. Relational Databases

The focus of this section is on the relational database model that is used in the implementation of the IOTS prototype database.

a. What is a Database?

A database is a structured collection of related data. A database is restricted to some implicit properties; it represents some real-world aspect, consists of a coherent collection of data that has inherent meaning, and is designed, built, and populated for a specific purpose. (Elmasri, 1994, p. 2)

b. Database Approach vs File Processing Approach

Database technology evolved to overcome the limitations of the traditional file-processing approach. In the file-processing approach, each user designs and implements the files needed for a specific application. In the database approach, the database forms a single repository that is defined once and accessed by multiple users each of whom may see a different view of the database. The difference between these approaches is important because neither end users nor programmers need worry about how data is physically stored in the database. This promotes application data independence wherein the database is managed by a Database Management System (DBMS) rather than the application programs.

The Database processing approach provides several advantages: it reduces data duplication, restricts unauthorized access, defines deduction rules in order to infer new information for the facts stored in the database, defines and enforces integrity

constraints, provides tools for backup and recovery, enables the Database Administrator to enforce standards, reduces application-development time, permits changes to the structure as the requirements change, and makes the data available to authorized users in an up-to-date state. (Elmasri, 1994, pp. 12-16)

c. Database Management System

A Database Management System (DBMS) is a set of programs that enables users or programmers to create and maintain a database. With the help of a DBMS, users are able to define the structure of a database, populate the database with data, and manipulate this data. A DBMS must provide enough functionality to manipulate both the database and its meta-data. The database and the software form together what is called a database system. Figure 10 shows a simplified database system environment (Elmasri, 1994, pp. 1-2)

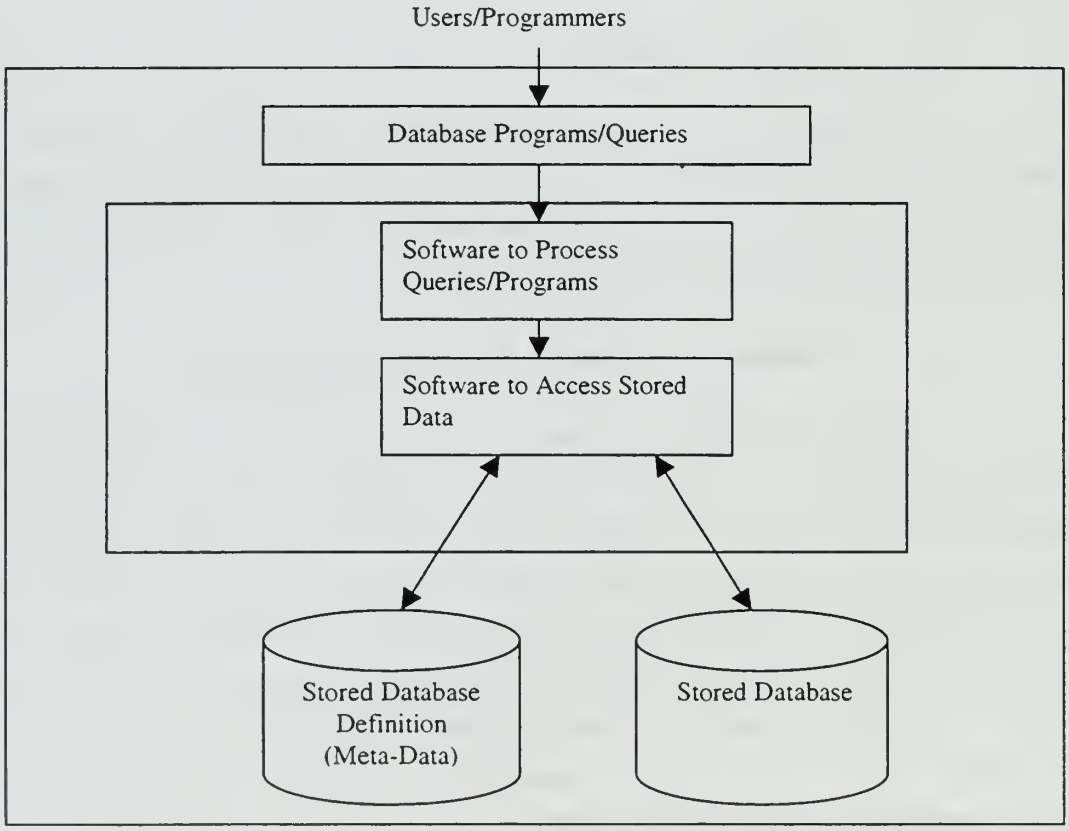


Figure 10. A Simplified Database System Environment (Elmasri, 1994, p. 2)

Accessing the database through the DBMS makes the application programs independent from the data. Application programmers do not have to worry about the ways in which data are physically stored in the database. At one time, there was a clear boundary between the application programs and the DBMS; programmers had to use a third generation language to access the database. Today, however, most DBMS have some extended capabilities to generate forms and reports that can ultimately be integrated in user's applications.

d. Relational Databases and Data Model

Database engineers define four different types of databases: Relational, Hierarchical, Network, and Object-oriented databases. The most common type in today's business world is the relational model. Figure 11 represents an example of a database maintained by a hypothetical company to track its employees and their dependents. The Figure shows the database structure and a few sample data records of such a database. Although relational databases faced some resistance in their early days, partly because of their high consumption of computer resources (Kroenke, 1998, p. 17), the relational model gained acceptance quickly in academic institutions because it is built on a provable mathematical foundation that hastened the process of developing a generalized query language (Taylor, 1999, p.24).

EMPLOYEE

Emp-No	Emp_Name	Date_Hired	Phone_Number	Monthly_Salary
1235	Mark Thomas	12-Feb-69	831-123-4567	$3000.00
1346	Maher Tony	23-Oct-80	408-675-8900	$5500.78
2385	George Watt	15-Jan-86	408-890-9999	$3500.59

DEPENDENTS

Emp_No	Dependent_Name	Sex	Date_of_Birth
1235	Susan	F	12-Nov-90
1235	Watt	M	23-Oct-97
1346	Kelly	F	28-Mar-98
2385	Omar	M	02-Jan-92
2385	Peter	M	23-Jun-95

Figure 11. A Database Example

In the relational model, data is stored in a certain way that minimizes duplicated data and eliminates certain types of processing errors. The relational model uses a process called normalization to break down undesirable tables into two or more desirable ones (Kroenke, 1998, p. 17). Several levels of normalization have been defined. Each level is used to examine the entity and its attributes to decide what attributes belong and do not belong to that entity.

"A data model is a set of concepts that can be used to describe the structure of the database." (Elmasri, 1994, p. 23) A relational data model represents the relational database as a collection of relations (tables). Each relation represents a data entity, and each entity is a collection of data attributes (columns). Relationships represent the interaction among these entities. An entity is an object about which we want to record information. An example of an object is an employee (see Figure 11). Possible attributes about this object are name, date hired, phone number, and monthly salary. In order to maintain *entity integrity*, each table must have a unique column value called a primary key. Any column that is a part of the *primary key* cannot be null. EMPLOYEE table uses Emp_No as a primary key whereas DEPENDENT table uses both the Emp_No and the Dependent_Name to uniquely identify a single record. Foreign keys are used to define relationships between relations. A foreign key in one relation is a reference to a primary key in another. An example of a foreign key in the DEPENDENT table is Emp_No, because it refers to a unique employee in the EMPLOYEE table. If a tuple (row) in the DEPENDENT table refers to another tuple in the EMPLOYEE table, the employee tuple must exist; this situation is referred to as *referential integrity*. A relationship can be one of these forms: One-to-One, One-to-Many, and Many-to-Many. "One-to-One" means for every occurrence of an instant of data in an entity, only one instance will occur in another. "One-to-Many" means for every single occurrence of an instant of data in one entity, many instances of data may occur in the other table. An example of this type is illustrated in Figure 3.2 where each employee in the EMPLOYEE table may have one or more dependents in the DEPENDENTS table. Finally, "Many-to-Many" means for multiple instances in a given entity, there are one or more instances in the related entity.

There are many high-level conceptual data modeling techniques that facilitate the database design process. One of these techniques is Semantic Object modeling which will be used in the next chapter in order to provide a high-level representation of the IOTS database.

2. Structured Query Language

IBM developed Structured Query Language (SQL) (pronounced SEQUEL) at their research facility. Nowadays, SQL is a standard in the relational database industry. Relational database users quickly learn this language because it expresses its logic with English-like syntax. SQL is a comprehensive database language with two types of statements: Data Definition Language statements (DDL) and Data Manipulation Language statements (DML). The DDL allows the creation of database tables and allows the creation of views on the top of these tables. The DML statements are used for querying and updating the database.

a. SQL Commands

A list of the most frequently used SQL commands and a brief description of each one of these commands is shown in Table 1.

SQL Command	Description
CREATE	Used to create tables
SELECT	This is the most frequently used command. Usually the SELECT command statement consists of three main clauses: SELECT, FROM, and WHERE. The WHERE clause is optional. Normally, the SELECT keyword is followed by a list of columns, the FROM is followed by a list of table names, and the WHERE is followed by one or more search conditions. Sometimes the keyword DISTINCT is optionally used after the SELECT command to remove any duplicate lines from the query results.
INSERT	Used to add a row into a table that already exists
UPDATE	Used to modify a record or more of data that meets a specified condition.
DELETE	Used to delete a row or more of data that meet the condition specified in the WHERE clause

Table 1. A List of the Most Frequently Used SQL Commands

Command	Syntax	Example
CREATE	**CREATE TABLE** <tablename> (col-def, …, col-def, tab-constr, …. tab-constr);	To create the EMPLOYEE table: **CREATE TABLE** EMPLOYEE (Emp_No **number**(4) **primary key**, Emp_Name **varchar2**(30), Date_Hired **date**, Phone_Number **char**(12), Monthly_Salary **number**(6,2));
SELECT	**SELECT** [distinct] <expression> **FROM** <tablename> [**WHERE** <search-condition>];	To list the numbers and the names of all the employees whose monthly salary is greater than $5000.00: **SELECT** Emp_No, Emp_Name **FROM** EMPLOYEE **WHERE** Monthly_Salary> 5000.00;
INSERT	**INSERT INTO** <tablename> [(column {, column})] **VALUES** (expression {,expression});	To add a new employee to the EMPLOYEE table: **INSERT INTO** EMPLOYEE(Emp_No, Emp_Name, Date_Hired, Phone_Number, Monthly_Salary) **VALUES** (1453,"Susi Kamil", 13 Nov 94, 831-642-8900, 6000.00);
UPDATE	**UPDATE** <tablename> **SET** <column>= <expression> … [**WHERE** <search-condition>]	To change the phone number for the employee named Maher Tony : **UPDATE** EMPLOYEE **SET** Phone_Number = 408-657-9342 **WHERE** Emp_Name = 'Maher Tony';
DELETE	**DELETE FROM** <tablename> [**WHERE** <search-condition>]	To delete the employee record for the employee with the employee number equal 1235: **DELETE FROM** EMPLOYEE **WHERE** Emp_No = 1235;

Table 2. Examples on the Most Frequently used SQL Commands

Table 2 shows the syntax of these commands in a very generic form, and presents an example of each command. These examples use the sample database shown in Figure 11.

b. How SQL is Used

Using SQL directly on top of a database is helpful in defining the structure of the database and providing quick *ad hoc* queries, but it is not very useful in building highly sophisticated applications. To build such applications, there are different ways to use SQL in order to manipulate the data in a given database, such as embedding SQL statements in a high-level third generation language. This is called Embedded SQL (ESQL). The ESQL statements are identified within the host language by marking their starting and ending points with special delimiters. Before compiling the host code with a native language compiler, a precompiler is used to convert ESQL statements into an equivalent host-language code. One major limitation to the ESQL is that the host program must know the type of the database it is going to connect to during the program development, and precompilers have been tied to particular database products (Orfali, 1998, p. 526).

Another way to execute SQL statements is called *stored procedures.* A stored procedure is a predefined and compiled procedure stored in the database that is available to be used by clients' applications and can be called by name.

Stored procedures have many benefits; after the first execution of the stored procedures there is no need to parse, optimize, or compile them again. The stored procedures are executed with a single sentence even though they may consist of multiple queries. Parameters are passed to a stored procedure enabling different clients to access the same stored procedure. Finally the stored procedures are fast in the client/server environment. Stored procedures, however, have some problems; they are vendor-specific, non-standard, and not portable across platforms. (Akbay,1999)

The following is a simple example on a stored procedure that is used to calculate the summation of the monthly salaries of all the employees listed in Figure 11. In practice, stored procedures can be used to perform more sophisticated tasks:

```
CREATE or REPLACE PROCEDURE sumSal (sal OUT NUMBER(8,2))
        AS
        BEGIN
        SELECT SUM(Monthly_Salary) INTO sal FROM EMPLOYEE;
```

```
END;
```

JDBC is another method used for sending SQL statements to relational databases. We defer the discussion of this method to the next section.

3. Summary

A database is a structured collection of related data. It evolved to overcome the limitations of the traditional file-processing approach. A database processing approach reduces data duplication, restricts unauthorized access, defines deduction rules, and enforces integrity constraints. A DBMS enables users to create and maintain a database. It makes application programs independent from the data. The relational model gained acceptance quickly because of its provable mathematical foundation. It represents the relational database as a collection of relations (tables). Each relation represents a data entity, and each entity is a collection of attributes.

The SQL is a standard in the database industry. It is easy to learn and use, and it consists of two parts: DDL and DML. The SQL can be used directly on top of a database, embedded in a high-level third generation language or as a predefined and compiled procedures stored in the database. Although stored procedures provide many benefits, they are vendor-specific, non-standard, and not portable across platforms.

B. JAVA AND JDBC

1. JAVA

Java is an object-oriented programming language developed by Sun Microsystems, which was formally announced in a major conference in 1995. Java attracted attention rapidly because it was designed for commercial business applications on the World Wide Web. Java applets are pieces of code that can be embedded in Web browsers. Applets can vary between little decorations and serious applications.

A major advantage of Java is its portability. Applications written in Java will execute on all major platforms. This cross-platform operability is provided by the Java Virtual Machine (JVM) that provides a run-time environment for running Java applications. Java source code is compiled into bytecodes, instead of machine code that is platform dependent. JVM executes these bytecodes directly or translates them to a

language understood by the machine running it. This feature leverages the desired cross-platform operability required by the IOTS.

a. Java Features

- **Garbage collection** - unlike other languages such as C++, Java takes care of allocation and deallocation of memory. Therefore, the burden of performing this task is no longer imposed on programmers. Unused memory resources that are no longer referenced are automatically removed in order to stop memory leaks that may cause applications to crash.

- **Extensibility** – Java enables programmers to extend the existing classes or write their own classes. The JDBC, explained later, is an example of Java extensibility. Other extension areas are multimedia, electronic commerce, and conferencing.

- **Simplicity** – Writing programs in Java is similar to C++, which makes it easy to learn for those who know C++ already. However, Java has eliminated the error-prone parts of C++, such as pointers, multiple inheritance, preprocessors, and operator overloading.

- **Strong typing** – Java enforces strong type checking. Therefore, many errors are caught at compilation time.

- **Network friendliness** – Java's built-in features facilitate client/server applications, and remote access to databases.

- **Scalability** – Java's scalability comes from its operating system independence. JVM enables Java-based applications to run on a wide range of platforms.

- **Multi-threaded** – Threads give Java programs the ability to do more than one function at the same time. This feature supports multimedia, and networking, due to the inherent slowness of network connections. To imagine this feature, think of a Java application that plays audio and displays text at the same time.

- **Security** – Java has some built-in security features. One feature is that JVM subjects any incoming code to its verifier that ensures the incoming code is correct, does not violate access restrictions, and does not forge pointers. Additionally, Java loader differentiates

between local class and classes imported from across the network. JVM does not allow the substitution of local classes by imported ones. Another important security feature is that JVM does not allow untrusted applets to access the local machine.

b. Java Development Environment

Typically, Java programs go through five phases to be executed. These phases are as follows: *edit*, *compile*, *load*, *verify*, and *execute*. Figure 12 (Deitel, 1998, p. 15) shows a typical Java environment.

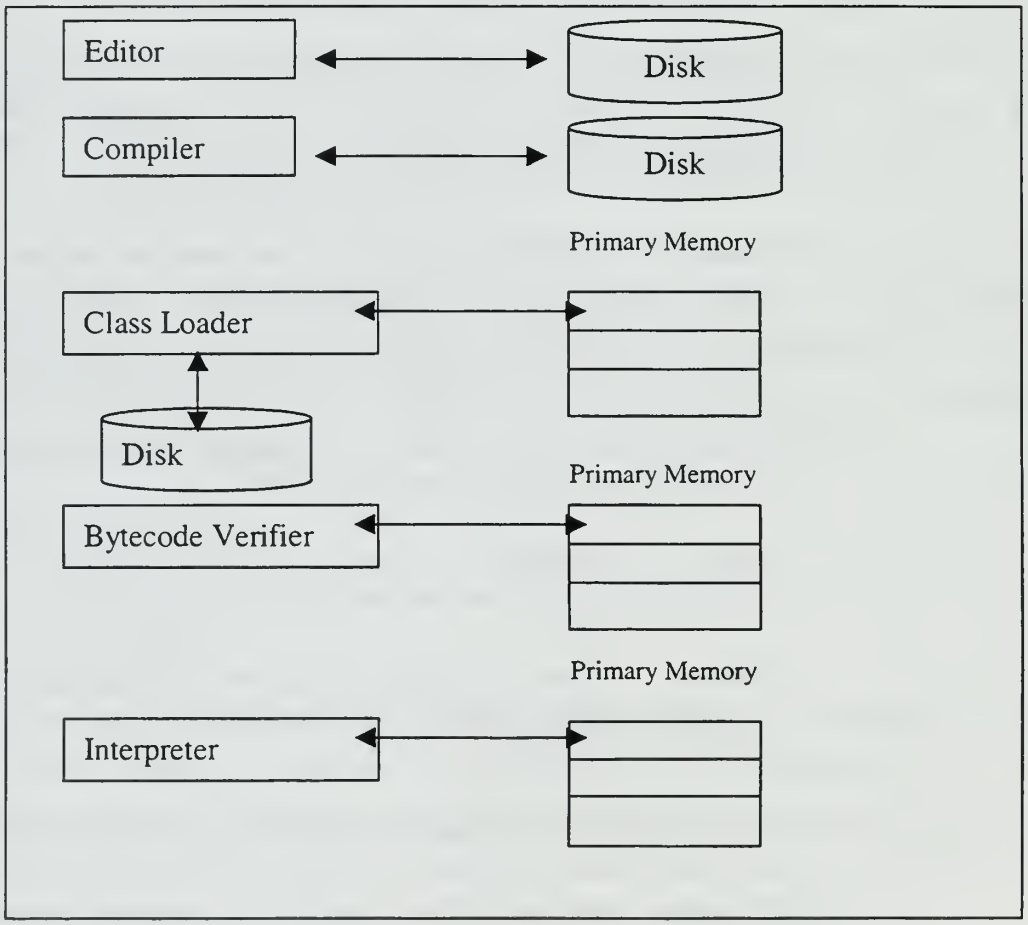


Figure 12. Typical Java Environment (Deitel, 1998, p. 15)

- **Editor** – enables programmers to create and edit Java programs.

- **Compiler** – generates the bytecode from the source code and stores it to disk.

- **Class Loader** – places bytecode in memory.

- **Bytecode Verifier** – ensures that the bytecode satisfies the Java security model.

- **Interpreter** – translates the bytecode into a machine language that the computer can execute.

2. JDBC

JDBC is a Java Application Programming Interface for executing SQL statements. JDBC is used to send SQL statements to relational databases. Thus, it is not necessary to write an application program to access each different database, such as Oracle, Informix, or Sybase. Rather, a single application program is written once using a JDBC Application Programming Interface (API) which will access the appropriate database by sending SQL commands. Java is secure, robust, and platform-independent whereas JDBC makes it easy to access relational databases by clients that are running on different platforms, for example, accessing a remote database from an applet running in a Web browser. JDBC makes it possible to derive the benefits of client/server systems and object-oriented languages yet still be able to access the legacy relational databases that businesses have huge investments in.

JDBC is a low-level API designed for building higher-level APIs that are more convenient and user friendly. There are two kinds of high-level APIs: (1) mix SQL statements directly with Java. For example, a Java variable can be used as a parameter in an SQL statement to pass or receive SQL values; (2) map the relational databases directly to Java classes; each table is mapped to a Java class, each row becomes an instant of that class, and each column value is mapped to an attribute of that instant. An example product of the first kind is JSQL and Java Blend is an example of the second kind. (Hamilton, 1997, pp. 6-7)

a. The JDBC Architecture

The JDBC architecture is best illustrated by Figure 13 (Orfali, 1998, p. 528). The Driver Manager is the backbone of the JDBC. Its main function is to connect the Java application to the suitable JDBC driver. The JDBC-ODBC bridge enables the JDBC to use Open Database Connectivity (ODBC). It was developed as a quick start to connect Java to SQL databases. The JDBC-ODBC provides access for databases when JDBC drivers are not implemented for them. ODBC binary code must be loaded on each client machine. Therefore, this kind of driver is appropriate for corporations for which installing client applications is not a problem. It is also appropriate on servers that are built in a three-tier architecture.

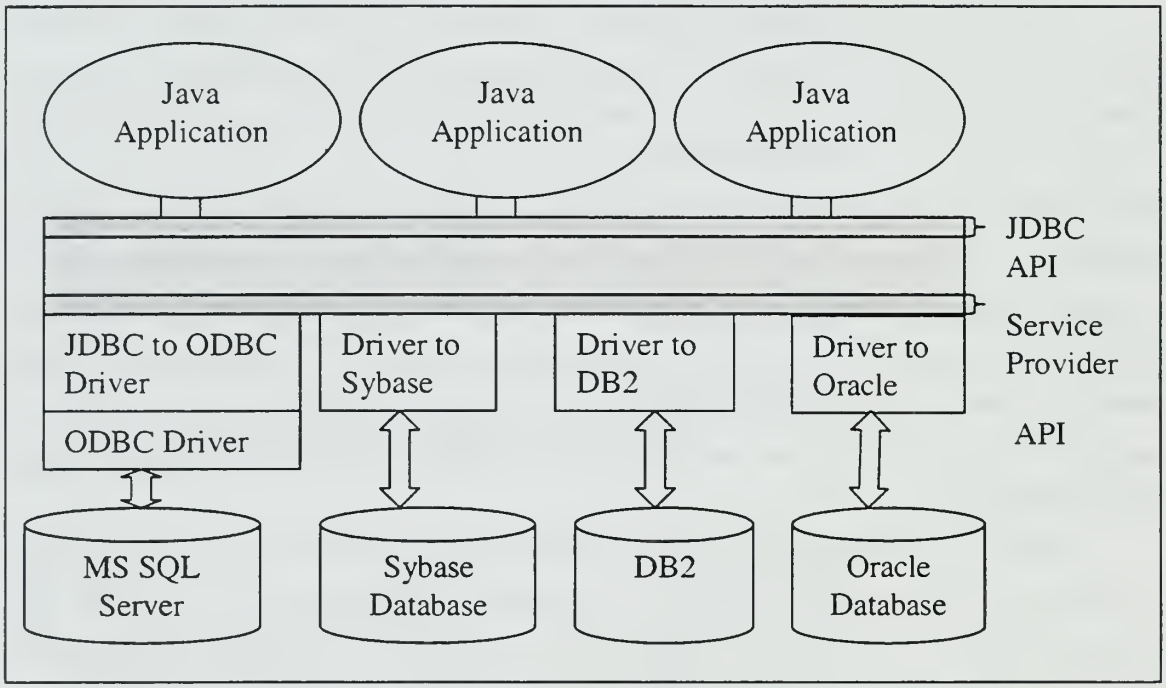


Figure 13. The Layers of JDBC (Orfali, 1998, p. 528)

b. JDBC Interfaces

Orfali and Harkey (Orfali, 1998, pp. 532-567) classify JDBC interfaces into the following four groups:

- JDBC core interfaces.

These are the interfaces and the abstract classes that must be implemented by every JDBC driver. These classes are basically used to locate the DBMS drivers, establish connections, submit SQL statements, and process the result sets. They perform over 90 percent of what you do with a database. Figure 14 shows these classes and interfaces.

- Java language extensions.

These are the new required extensions for SQL, because the JDBC requires dealing with high-precision data types that are mostly used to process data representations of money. SQL also defines its own exception and warning types that should be grafted into Java. Figure 15 shows Java.sql extensions for Java.lang.

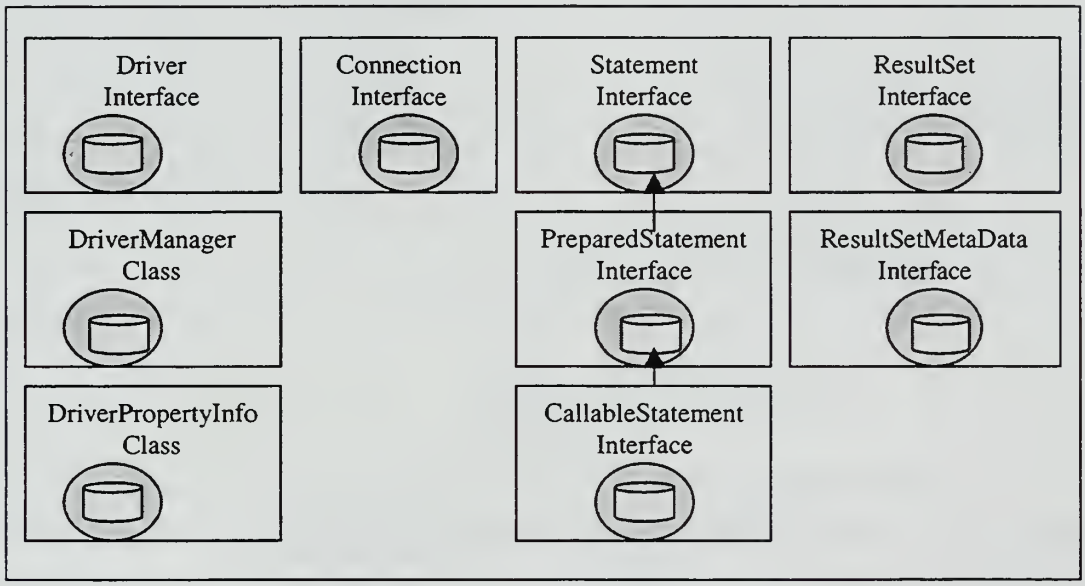


Figure 14. The JDBC Core Classes and Interfaces (Orfali, 1998, p. 533)

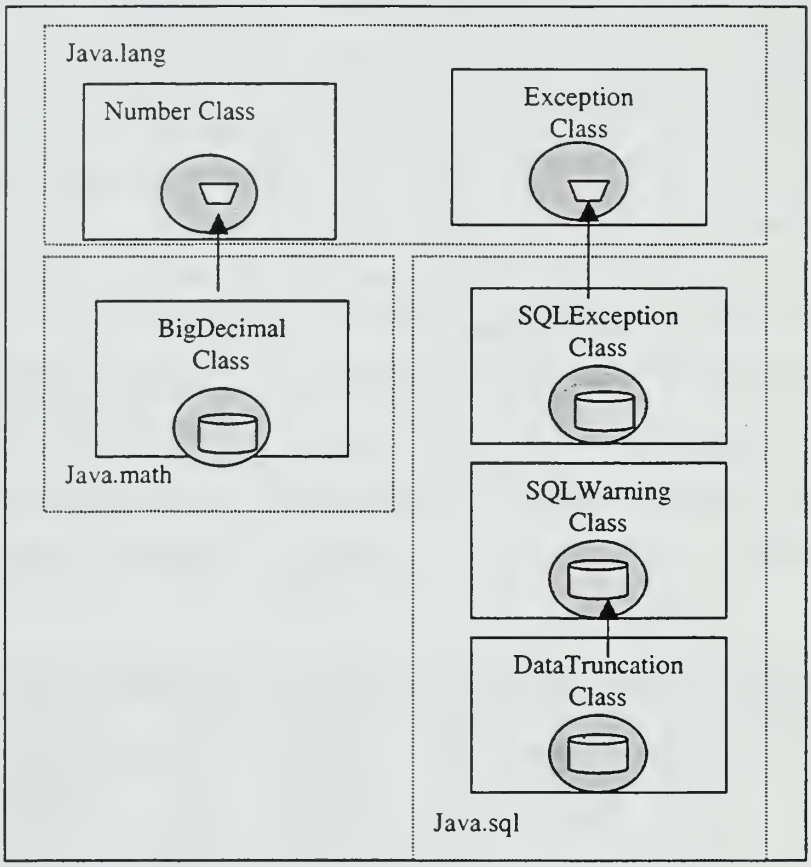


Figure 15. Java Language Extensions (Orfali, 1998, p. 535)

- Java utility extensions.

Java extends the Java.util.Date class to get three new classes defined in Java.sql in order to meet the requirements of having fine-grained time and date utilities. These new classes allow the user to measure the time in nanoseconds. These classes are shown in Figure 16.

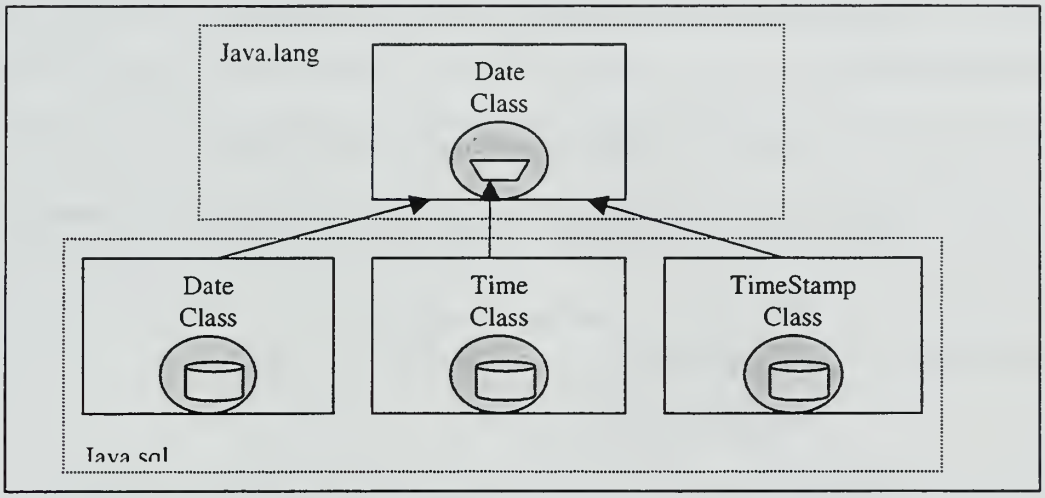


Figure 16. Java Utility Extensions (Orfali,1998, p. 536)

- SQL metadata interfaces.

This interface standardizes the access to the database metadata across multiple DBMS vendors. This interface consists of 133 methods that should be implemented by JDBC vendors.

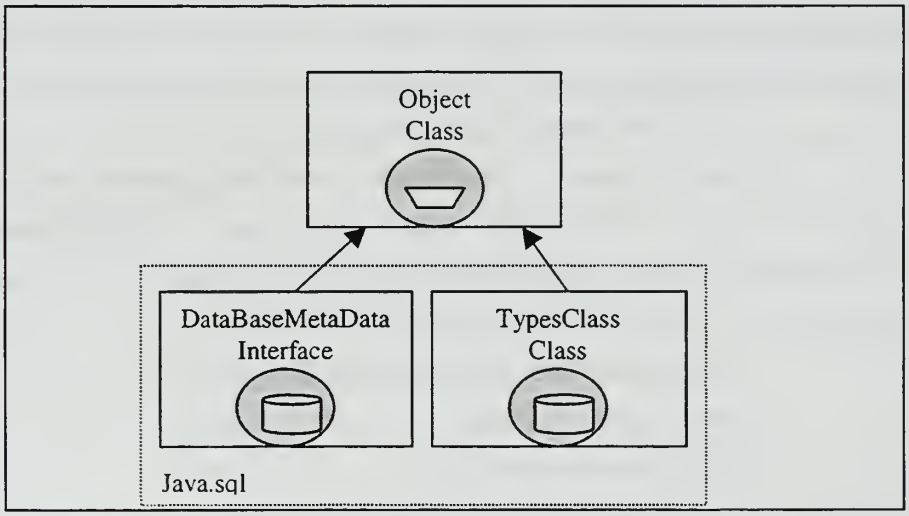


Figure 17. The Java.sql Metadata Interface and the Types Class(Orfali, 1998, p. 537)

Figure 17 shows the Java.sql Metadata Interface and the Types class. The Types class is just a placeholder for constants and does not support any methods. (Orfali, 1998, PP. 532 – 567)

c. Accessing a Relational Database with JDBC

With JDBC, performing the following tasks is easy:

- Establish a connection to a database.
- Send SQL statements.
- Process the results.

The following is a brief description of the above JDBC capabilities:

(1) Establishing a connection with the database

To perform this task, the driver must be loaded first. Then a connection is made. For example, to load a JDBC-ODBC driver, the following single line of code might be used:

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
```

There might be a need to refer to driver documentation to get the class name of that particular driver. For example, if a class name for a driver is Jdbc.DriverX, then you would use the following statement to load the driver:

```
Class.forName ("Jdbc.DriverX");
```

Once the driver is loaded, the connection can be established. For example, if you are using a JDBC-ODBC driver to connect to a Microsoft Access database, with a source name = IOTS_DB, user name = "guest", and a password = "Moon", you would use the following statements:

```
String url = "jdbc:odbc:IOTS_DB";
String login = "guest";
String password = "Moon";
Connection con = DriverManager.get Connection (url,login,
password);
```

You may need to refer to the documentation of the driver's vendor in order to figure out the syntax of the URL parameter in the getConnection method.

(2) Send SQL Statements

In order to execute SQL statements on the database, we use the connection object created already in the previous step to create a statement object. If the SQL statement intends to query the database, the executeQuery method of the statement object is executed; otherwise if the SQL statement attempts to create or modify a table of the database, the executeUpdate method is used instead. For example, to create the ORACLE EMPLOYEE table listed in Figure 11, the following set of statements are used:

```
String createStatement = "CREATE TABLE EMPLOYEE ("
+ "Emp_No number(4) primary key, " +
"Emp_Name  Varchar2(30), " +
"Date_Hired date, " +
"Phone_Number  char(12), " +
"Monthly_Salary  number(6,2)
Statement stmt = conn.createStatement();
stmt.executeUpdate(createStatement);
```

(3) Processing the results

The results of a SELECT statement are returned in a ResultSet object. This object provides a set of useful methods to manipulate the returned object. One of these methods is the next() method. This method moves the cursor form the current row of the ResultSet object to the next row. The following statements would be used to list the numbers and the names of all the employees whose monthly salary is greater than $5000.00:

```
String query = "select Emp_No, Emp_Name " +
"from EMPLOYEE " +
"where Monthly_Salary> 5000.00 ";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(query);
System.out.println("Employee Number     Employee Name";
System.out.println("----------------------      --------------------";
while (rs.next() ) {
Integer empno = rs.getInt("Emp_No");
```

```
String name= rs.getString("Emp_Name");
System.out.println(empno + "    " + name);
}
```

d. JDBC and the Client/Server Models

The JDBC API supports database access utilizing both two-tier and three-tier client/server models. What decides the client/server tier level is the way in which major functional components of an application are split between the client and one or more of the servers. The typical functional units are the user interface, the business logic, and shared data. Two-tier client/server systems have the business logic either inside the user interface on the client, or the database server (or both); whereas three-tier client/server systems have the business logic on the middle tier separate from both the user interface and the data. Three-tier client-servers systems are more scaleable, robust, and flexible. Moreover, they can integrate data from various sources. (Orfali, 1998, p. 583)

(1) JDBC Two-tier

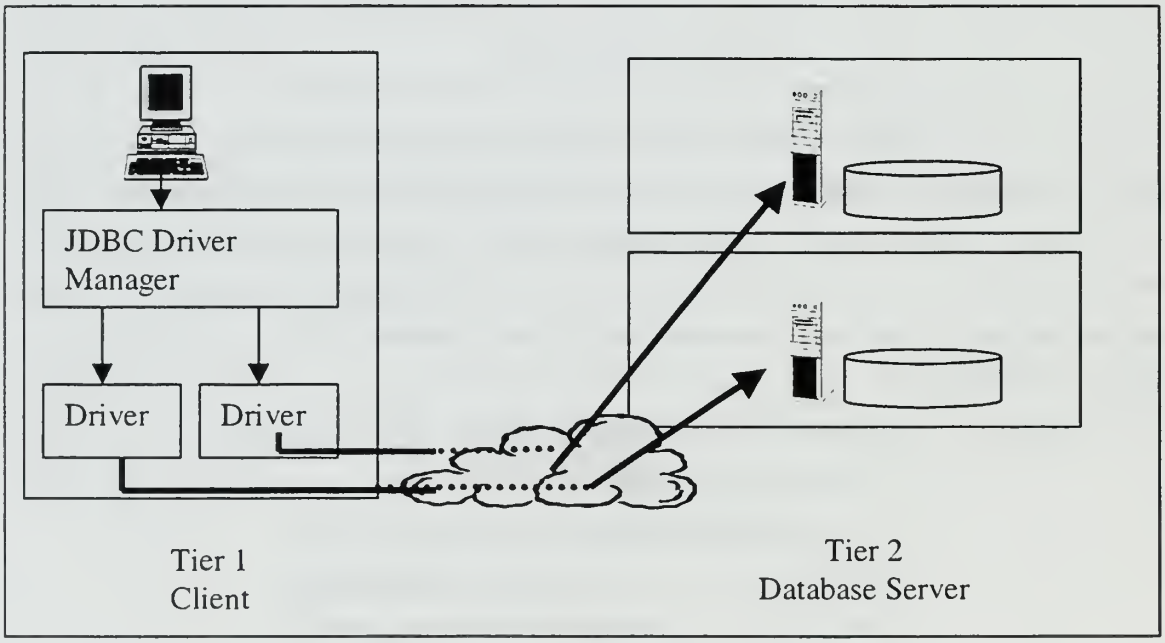


Figure 18. JDBC Two-tier (Orfali, 1998, p. 584)

This model is called the "fat client" approach since the client holds the business logic and maintains all the JDBC drivers. Figure 18 shows a two-tier client/server split.

The load on the client can be reduced by using stored procedures that lie on the database. However, the stored procedures are non-standard and the user might be locked into a proprietary vendor implementation. To achieve portability, stored procedures should not be used.(Orfali, 1998, p. 583)

(2) JDBC Three-tier

This approach moves the business logic from the client to the middle tier. Applications that access the data run on the server side in addition to the DBMS engines. The databases can be run on separate server machines. Figure 19 shows the JDBC three-tier client/server split. (Orfali, 1998, p. 586)

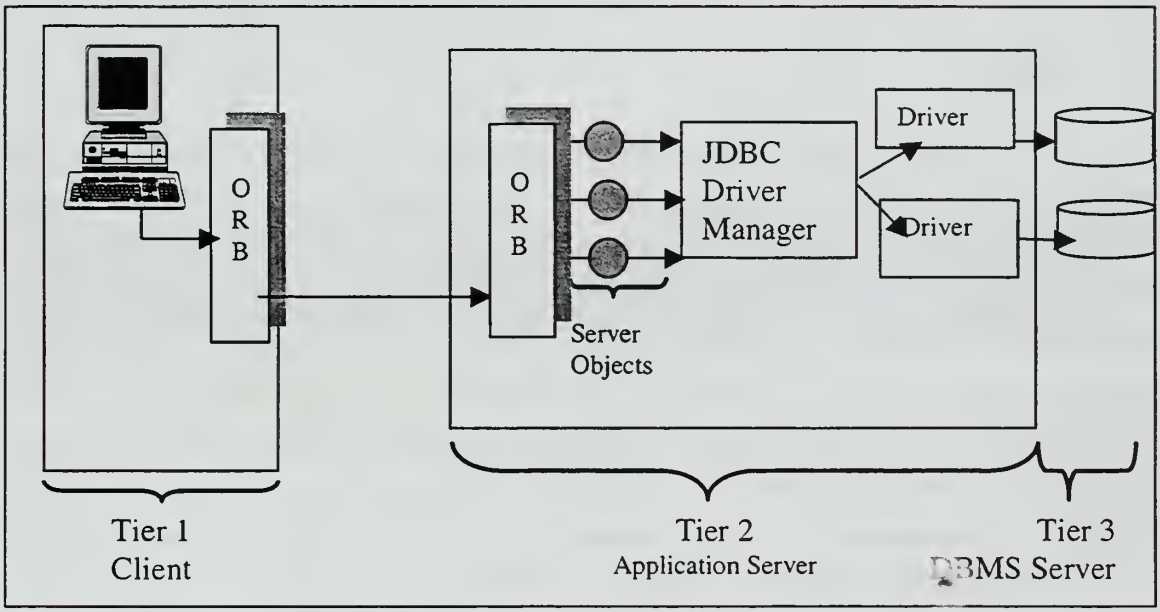


Figure 19. Three-tier (Orfali, 1998, 586)

The above figure shows that the Object Request Broker (ORB) is used as a communication backbone between the client and the application server. We describe the ORB in much more detail in the next section. The three-tier model is attractive because the middle tier makes it possible to maintain control over access and

types of updates that can be made to the data. Only the middle tier knows how to find and manipulate data. So, the middle tier can be thought of as an intermediate security level that shields the client from direct access to the database. Moreover, the clients can be designed to use a higher level API that can be translated into a low-level call by the middle tier.

3. Summary

Java is an object-oriented programming language. It has attracted attention mainly because of its features such as portability, garbage collection, extensibility, simplicity, strong typing, network friendliness, multi-threading, and security. Typically, Java programs go through editing, compiling, verifying, and executing phases. JDBC is a low-level API that establishes a connection with a database, sends SQL statements, and processes the results. The JDBC supports database access utilizing two-tier and three-tier client/server models. The three-tier architecture makes the application more secure, scalable, and manageable.

C. CORBA

The Common Object Request Broker Architecture (CORBA) is used by a wide spectrum of the computer industry for creating distributed object systems. Object Management Group (OMG), an international non-profit consortium that includes more than 800 company members from all over the world, sets CORBA standards and specifications. One third of these members come from outside the United States. Microsoft is not a member of this consortium because it has its own proprietary product for distributed object systems called *Distributed Component Object Model* (DCOM). OMG does not produce software; it only defines specifications and makes them freely available for implementation to any interested party. Member companies meet frequently to reach a consensus on the specifications. Software that adheres to these standard specifications can interoperate with other software that meets the same standard.

CORBA objects can live anywhere on a network, where they can discover each other and interoperate on the object bus. Remote clients can access these objects via method invocation. The location of any CORBA object is transparent to the client; these objects can be in the same process or sit on some other machine across the network. The client does not need to know how the server object is implemented. What the client needs

to know is only the published interface to that object. Therefore, the client might be implemented in Java while the server is implemented in some other language, such as C++, Smalltak, or Ada. The interface is written in a neutral Interface Definition Language (IDL) that defines a distributed object's boundaries. IDL makes all the objects that reside on the CORBA bus operating system and language independent. (Orfali, 1998, pp. 3-6)

1. CORBA Architecture

The OMG first published the Object Management Architecture Guide (OMA Guide) in 1990. It was revised in1992 with additional details added in 1995.

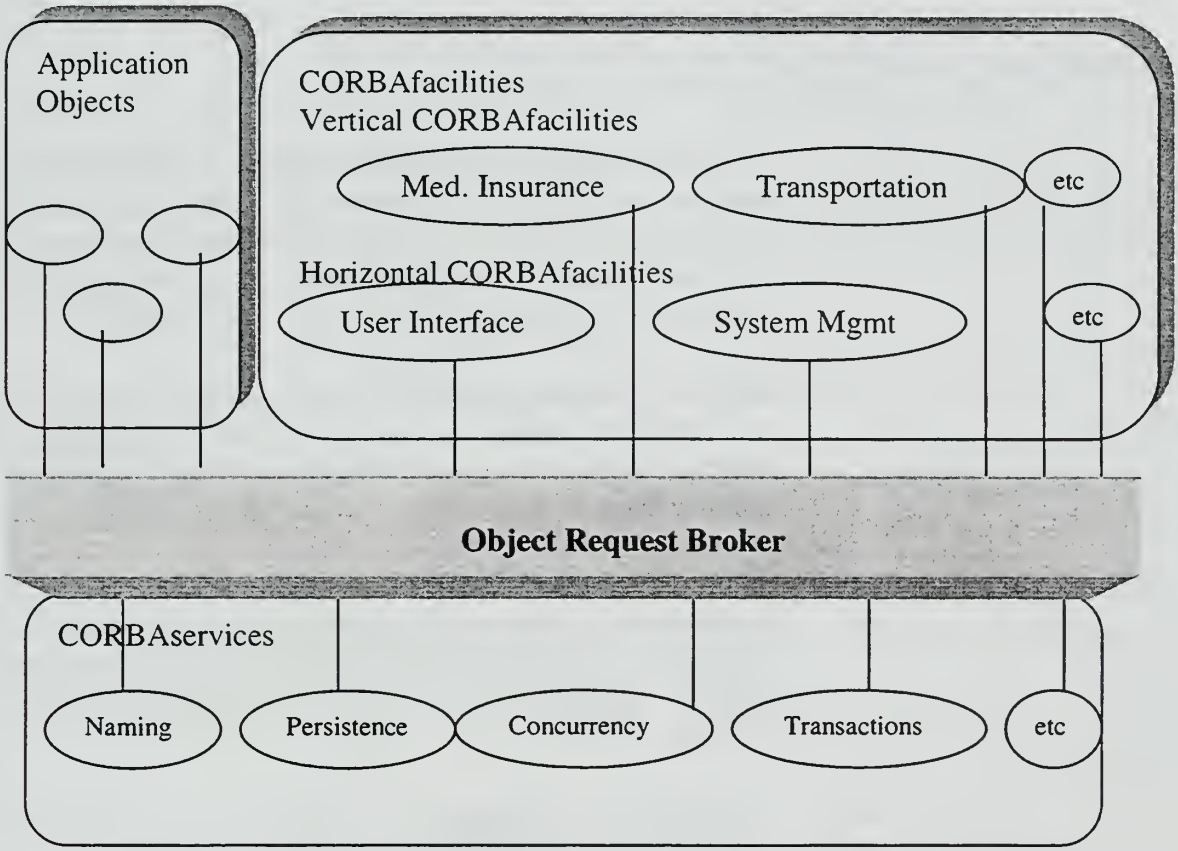


Figure 20. Object Management Architecture

Figure 20 shows the four major parts of the architecture:

- Object Request Broker.
- CORBAservices.
- CORBAfacilites.
- and Application Objects

a. The Object Request Broker (ORB)

The ORB forms the object bus that enables objects to transparently send requests to-and receive responses from–other objects located locally or remotely. The ORB enables objects to discover each other at run time and invoke each other's services. "In theory, CORBA is the best client/server middleware ever defined. In practice CORBA is only as good as the products that implement it." (Orfali, 1998, pp. 7-8)

(1) ORB Benefits

According to Orfali and Harkey (Orfali, 1998, pp. 8-9), the CORBA ORB provides the following list of benefits:

- **Static and dynamic method invocations.** A CORBA ORB provides the ability to either statically define the method invocation at compile time, or dynamically discover them at run time. Static invocation provides strong type-checking whereas dynamic invocation provides more flexibility.

- **High-level language bindings**. This benefit provides the user with the means to use a high-level programming language of his choice, yet the user is able to access server objects, regardless in which programming language the objects are written.

- **Self-describing system**. The ORB maintains an Interface Repository that contains the description of all the functions a server provides. Clients use this metadata to generate code "on-the-fly".

- **Local/remote transparency**. An ORB can run on a single machine or on multiple machines interconnected with CORBA Internet Inter-ORB Protocol (IIOP). In either case, a CORBA programmer does not have to be concerned about "transports, server locations, object activation, byte ordering across dissimilar platforms, or target operating systems – CORBA makes it all transparent"(Orfali, 1998, p. 9).

- **Built-in security and transactions**. The ORB includes context information to handle both security and transactions.

- **Polymorphic messaging.** a method invoked on objects has different effects depending on the object receiving it .

- **Coexistence with existing systems.** Using CORBA IDL, the user can wrap existing code to look like an object no matter how it was implemented. (Orfali,1998, pp. 7-9)

(2) The Structure of CORBA 2.0 ORB

The core ORB provides the facilities required to write distributed applications using different programming languages and operating systems. The basic function of the ORB is to provide the client with the facilities required to send requests to server objects and to get responses back. Figure 21 shows client and server interaction through the ORB. The following is a brief discussion of the various parts that comprise the ORB.

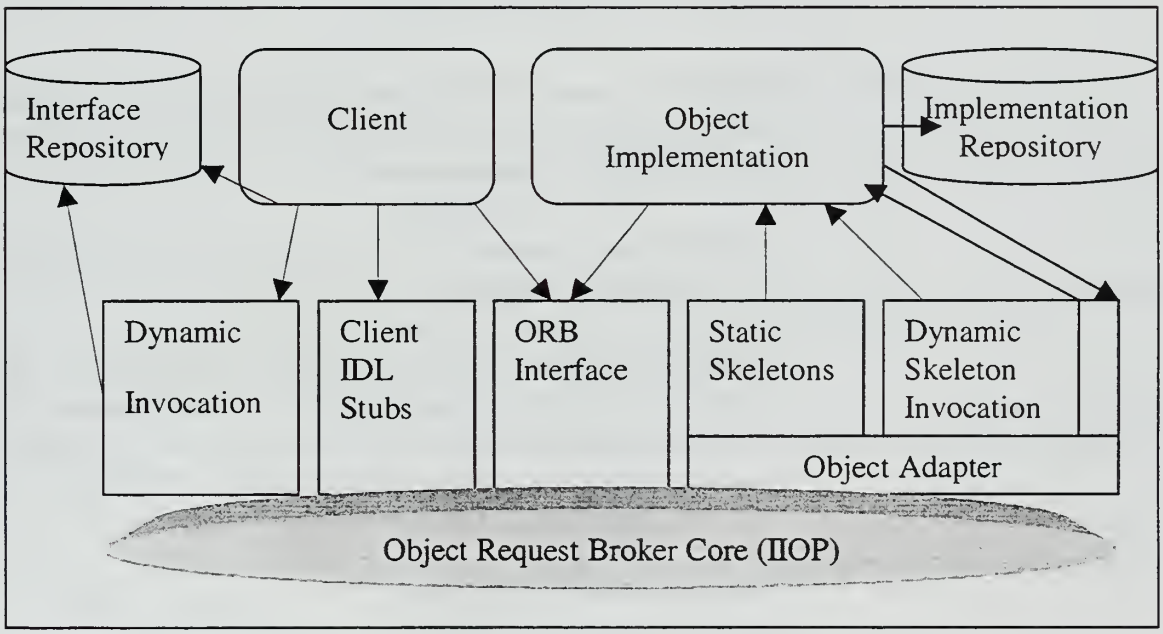


Figure 21. The Structure of CORBA 2.0 ORB (Orfali, 1998, p.11)

- **Client IDL Stubs**: These stubs provide the client with the required interface to invoke statically services on the corresponding servers. IDL is used to generate both client and server stubs.

- **Dynamic Invocation Interface (DII):** A client can dynamically access the metadata in the Interface repository, discover its methods, and invoke these methods at run time. Static invocation provides strong type-checking at compile time which helps in discovering errors during development and making the application more robust, whereas the DII provides more flexibility.

- **Interface Repository APIs**: The ORB is a self-describing bus which maintains a repository that contains the descriptions of all registered interfaces. Clients mainly use this repository for dynamic invocation of methods on object servers.

- **ORB Interface**: This consists of APIs to some services that can be used by an application, such as converting an object reference to a string.

- **Object Adapter**: Provides an environment to instantiate server objects, assign them object IDs, and register them with the implementation repository.

- **Server Static Skeleton**: Provides an interface for statically accessing server objects.

- **Dynamic Skeleton Interface (DSI)**: Corresponds to the client's DII. These services handle clients' requests by finding the objects that offer their requests. DSI is useful in helping differently implemented ORBs that have no previous knowledge of each other, to interoperate.

- **Implementation Repository**: This repository consists of the information about the objects a server supports.

(3) General Request Flow

Clients make requests and the server objects act on them. Henning and Vinoski describe the request flow from the client application, through the ORB, and up to the server application in the following manner (Henning, 1999, pp. 16-17):

- The clients either make a request using static stubs or using the DII. Either way, the request is directed into the ORB.
- The ORB on the client side sends the request to the ORB on the server side.
- The ORB core on the server dispatches the request to the object adapter.
- The object adapter further dispatches the request to the servant that implements the target object.
- After the request is executed, the servant returns the response to the client.

There are different styles of CORBA requests (Henning, 1999, p. 17):

- A *synchronous* request: the client blocks while waiting for a response.
- A *deferred synchronous*: the client sends a request, continues processing, and later polls for the response.
- A *oneway* request: the client sends a request that may not be delivered to the target object and no response is allowed.
- Future version *asynchronous* requests: allows the clients and server who are occasionally connected to communicate with one another.

b. CORBAservices

CORBAservices are useful system-level utilities that augment or complement the functionality of the ORB. These services provide the application

developer with the ability to manage, store, create, name, and locate objects. The following is a list of the more important services:

- **Naming Service**: provides the ability to find an object, given its name.

- **Event Service**: allows a client or a server to send a message, or event, to any number of receivers that registered their interest in specific events. This service introduces the notion of event channel; an event generator sends events to this channel and the event receiver receives the events from this channel.

- **Security Service**: provides a framework to protect objects or groups of objects. The service addresses identification and authentication, confidentiality, and non-repudiation.

- **Concurrency Service**: provides a locking mechanism to control the access to an object on behalf of either transactions or threads.

- **Property Service**: provides operations to associate properties with any object.

- **Transaction Service(OTS):** provides two-phase commit coordination to control the commitment and abortion of transactions. These transactions may span multiple databases, of the same or of the different types.

- **Trading Service.** This is an enhancement of the naming service which allows objects to be registered with a set of properties and then searched for by specifying constraints over these properties.

- **Relationship Service**: allows the construction and management of relationships between objects that know nothing of each other.

- **Query Service**: allows query operations to be executed against a collection of objects. This service is based on SQL3 and OMG Object Query Language (OQL).

- **Persistent Objects Service**: defines a framework for how the communication between a database and an object should be. It deals with storing and restoring this object to and from the database.

- **Externalization Service**: provides a mechanism for copying an object to another location. Externalization means recording the state of an object in a stream of data that can be saved or transferred across the network. Reversing the process, which is called *internalization*, restores the externalized object.

- **Life Cycle Service**: provides interfaces that allow objects to be created, moved, copied, and deleted from the ORB.

- **Licensing Service**: allows a server to determine whether a machine on which it is running is licensed to run the software or not.

c. CORBAfacilities

Whereas CORBAservices provide utilities for individual objects or groups of objects, CORBAfacilities provide higher level support for applications. There are two types of CORBAfacilities: horizontal CORBAfacilities which can be used by virtually every business, and vertical CORBAfacilities that standardize some specific business areas.

2. GIOP and IIOP

General Inter-ORB Protocol (GIOP) defines a standard for sending ORB requests over some low-level communication protocol. GIOP was designed to work over TCP/IP, but could be used over Novell's IPX or a future protocol. The Internet Inter-ORB Protocol (IIOP) is the implementation of GIOP over TCP/IP. IIOP makes CORBA 2.0 ideally suited for Internet–and Intranet–based applications. According to the CORBA 2.0 specification, every ORB must implement the Internet Inter-ORB Protocol (IIOP) or provide a half-bridge to it. A half-bridge is a module that translates between the internal protocol and the IIOP. This requirement ensures that ORBs from different vendors are able to interoperate with each other.

3. Interface Definition Language

Interface Definition Language (IDL) is a declarative language-neutral standard language used to define a CORBA object interface via which an object advertises its services. An IDL interface provides the ORB with the information required to pass

messages to and from the object and a transparent access to the object. Language independence gives application developers the flexibility to implement their applications with the programming language that suits their requirements. IDL provides the ability to use different languages in the same system. Moreover, existing legacy code can continue to be used by creating an IDL wrapper for it. This wrapper will make the legacy code appear as a distributed component on the object bus. A number of IDL mappings exist to some languages, such as C++, Ada, and Java.

a. The Structure of IDL

IDL separates the specification of an object from its implementation. It is used to define an object's attributes, services, parameters, types, and exceptions. Figure 22 shows the structure of an IDL file.

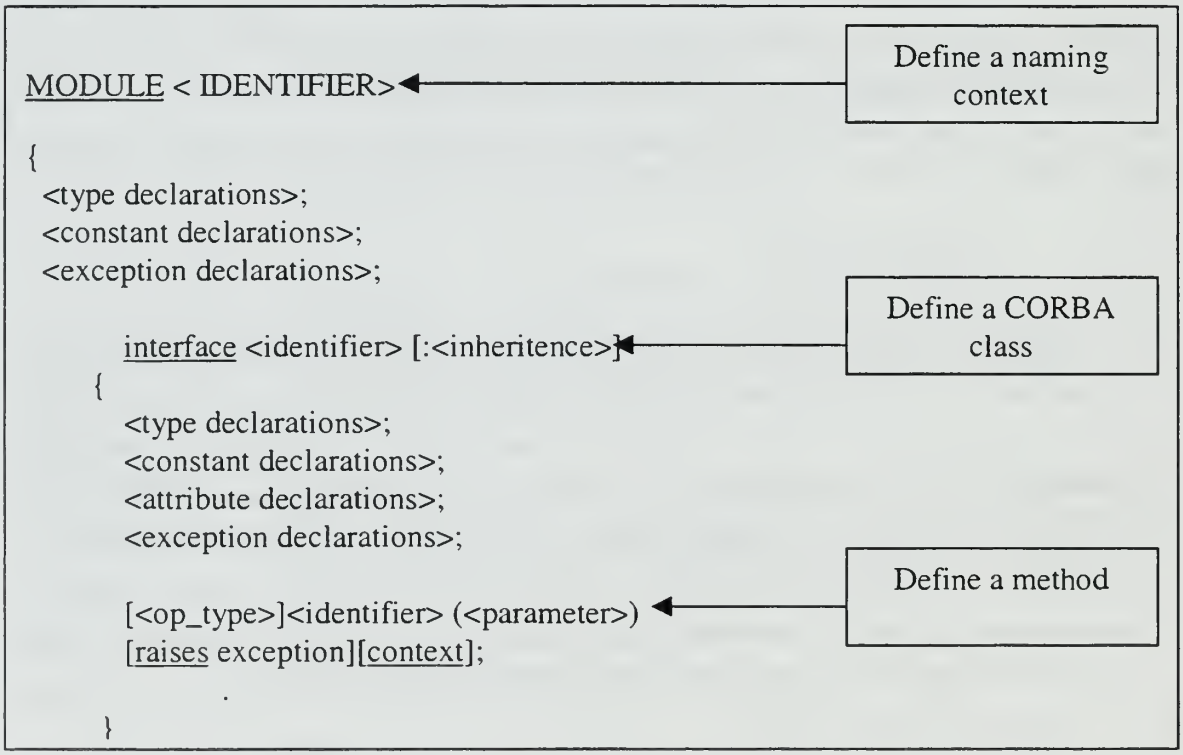


Figure 22. The Structure of a CORBA IDL File (Orfali, 1998, p. 420)

The following is a brief discussion of the major elements that constitute a CORBA IDL file:

- **Modules**: a module is a construct that provides a namespace that groups logically related IDL definitions. This construct is similar to

a package in some other languages; for example, in Java, once an IDL file is compiled a module is mapped to a Java package. An IDL module introduces an additional level of hierarchy in the IDL namespace.

- **Interfaces**: It defines a set of operations that can be performed on an object. An interface is similar to a class definition, but without an implementation. IDL interfaces support multiple inheritance and can declare one or more exceptions.

- **Operation**: CORBA IDL defines the methods that a client can invoke. A method definition, a signature, consists of the method's parameters and the method's return type.

- **Data types**: The accepted values of CORBA parameters, exceptions, attributes, and return values.

IDL Type	**Definition**
Short	$-2^{15} .. (2^{15} -1)$
Unsigned short	$0 .. (2^{16} -1)$
Long	$-2^{31} .. (2^{31} -1)$
Unsigned long	$0 .. (2^{32} -1)$
Float	IEEE single precision
Double	IEEE double precision
Char	ISO Latin-1 character for graphic characters, ASCII for NULL and formatting characters
Boolean	TRUE or FALSE
Octet	8-bit uninterpreted
Any	Expresses any IDL type

Table 3. IDL Types (lewis, 1998, p. 61)

There are two categories of data types: basic built-in types and constructed types. IDL uses its basic built-in types and its constructs for virtually creating any desired data type.

b. Basic Built-in Types

IDL's built-in types are showed in Table 3.

c. Constructed Types

IDL defines some structured data types including strings, enumerated types, structs, discriminated unions, and sequences. For more details on these data types the reader may consult one of the IDL books listed in the references.

4. CORBA/Java and the Web

The Internet has created the potential for a high level of business growth because of its reachablility to new customers and for its drastic cost reduction of information distribution. The Internet makes it possible for suppliers and consumers to be located worldwide. Nowadays, the Internet is one of the major communication tools that connect individuals, businesses, educational, and governmental organizations.

There are different models to develop multi-tier client/server applications that take advantage of the potential provided by the Internet. Some of these models are the following: HTTP, HTTP with CGI, and CORBA. HTTP–based client/server interactions are stateless; the client establishes a connection to the server, makes a request, waits for a response, and then the connection is closed. HTTP is designed mainly to distribute static content to a large audience. The Common Gateway Interface (CGI) makes the HTTP servers dynamic. CGI launches a new process to service each incoming client request. Thus, HTTP with CGI is slow and stateless which makes it unsuitable for distributed object applications. Many vendors attempt to get around the stateless nature of HTTP and CGI by providing some extensions, but some of these extensions are vendor-specific and platform-dependent, which does not naturally fit the heterogeneous nature of the web.

Many applications require context state to be maintained across multiple client requests. CORBA clients have a fine-grained control over the server objects. Whereas the HTTP clients cannot invoke methods directly on the server object, the CORBA clients can. Moreover, CORBA clients can pass a large number of parameters and user-defined objects to the server objects. HTTP clients always instantiate a request and there is no means for the servers to call clients and notify them with asynchronous events. CORBA supports callbacks in which a client passes an object reference to the server and the server

later can invoke an operation on that object, so client and server objects can have a peer-to-peer relationship.

Appendix E shows Orfali and Harkey's rating for the major six Object Web technologies including CORBA/IIOP, Microsoft's Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), HTTP/CGI, Servlets, and Sockets. For more details, the reader may consult with Orfali and Harkey's book listed in the references. (Orfali, 1999, p. 375)

5. Summary

CORBA is used for creating distributed object systems. CORBA objects can live anywhere in the network, where they can discover each other. CORBA clients and servers can reside on different platforms, operating systems, and networks and still be able to interoperate as long as they implement the same IDL. CORBA overcomes the stateless nature of the HTTP by enabling clients to have a fine-grained control over the server objects. CORBA objects have a peer-to-peer relationship where both client and server objects can invoke methods on each other.

After this brief overview of the technological background material, the reader will be ready to read the design, implementation, and analysis of the IOTS prototype in the next three chapters. Readers seeking more details about this technology may consult one or more of the references provided.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IOTS RELATIONAL DATABASE DESIGN AND IMPLEMENTATION

In this chapter, we build the IOTS data model using the Semantic Object Modeling technique. Next, we transform this model into a relational database design. Finally, we implement our design and generate the database schema. This chapter consists of the following sections:

- Development methodology
- Proposed relational model
- Database implementation

A. DEVELOPMENT METHODOLOGY

The methodology advocated for the development of the data architecture of the IOTS is a three-step process. As shown in Figure 23, the steps of this process are as follows: 1) define objects/entities, attributes, and relationships, 2) develop the data model, and 3) transform the data model into a relational database schema. Although these three steps are listed in a chronological order, there is no hard line that separates any two consecutive steps. The further we go in the development process, the we understand the requirements. Therefore, to get a satisfactory product, there is always a need to refine within our budget for resource usage and to consider the return of investment.

The system requirements, discussed in Chapter II, are the main source of input to developing the relational database model. Theoretically, other major sources of information can be used, such as the following: 1) existing documentation, screens, reports, data files, 2) questionnaires, and 3) interviews with developers, programmers, administrators, and end-users. In this research, we build our model based on the following inputs:

- The underlying assumptions about the system and the research's objectives discussed in Chapter I.

- The system requirements, as listed in Chapter II.

- Our experience in developing and managing Inventory Ordering and Tracking Systems. This previous knowledge helped us achieve our goal of designing a modern IOTS prototype. We employed some of the most common business rules used by some of the inventory ordering and tracking systems in today's business. We also utilized our knowledge of

existing legacy reports, forms, screens snap shots, and data files. However, in the case of developing a system that is not constrained to our underlying assumptions, the developers' judgement and experience in the system under design will never replace the real stakeholders' points of view. This judgement and experience would rather provide a complementary perspective.

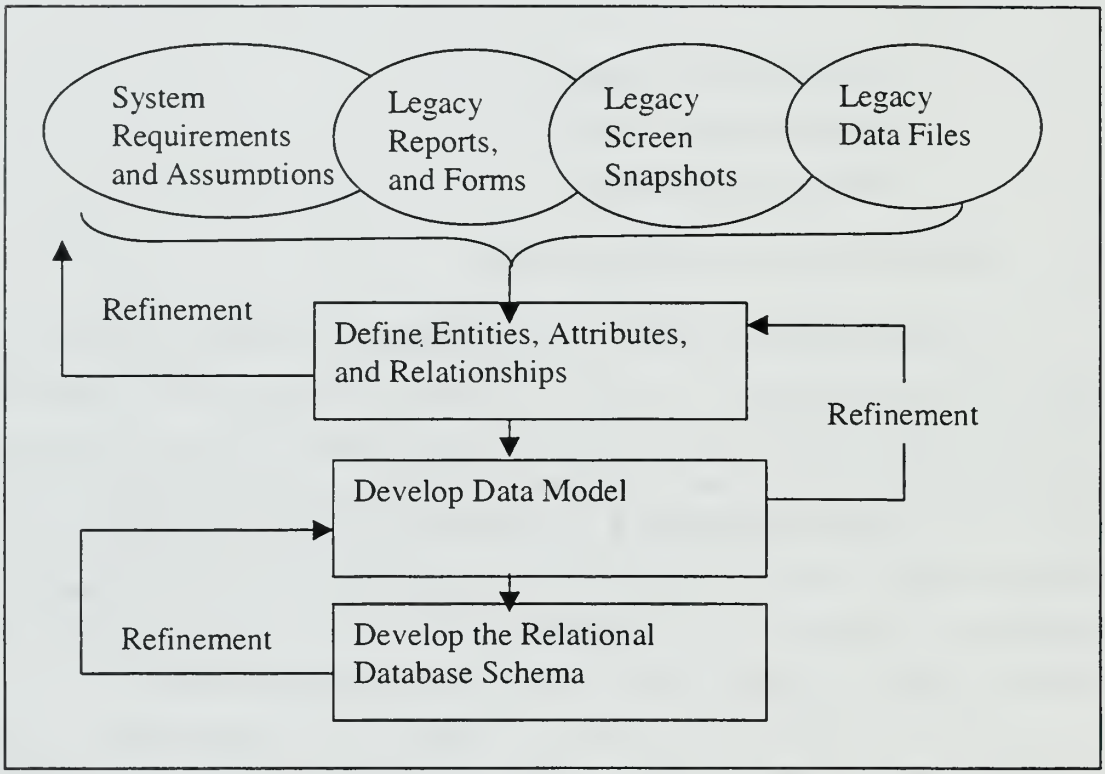


Figure 23. Data Model Development Methodology

B. PROPOSED RELATIONAL MODEL

In this section, we follow the methodology discussed above in order to develop the relational database schema.

1. Define Entities, Attributes, and Relationships

The first step in the development of the IOTS database model is to discover the candidate entities. These entities, at the enterprise level, are referred to as *data subjects.* The data subjects for the IOTS are listed and defined in Appendix F. Defining these data subjects is important to the development process. It is critical that all personnel involved

in this process agree on these definitions in order to avoid any possible future inconsistencies between the data model and the IOTS prototype. In order for the prototype to meet the stakeholders' expectations, it must be built on a well-defined basis.

After the data subjects were defined, a list of candidate attributes was defined and showed in Appendix G. Appendix H shows which attributes belong to which data subject. Data subjects that are marked as "Independent" can exist on their own, whereas "Dependent" data subjects will exist only if their parents exist.

Finally, the relationships between the data subjects were identified and defined as shown in Appendix I. Defining the relationships between these data subjects is a necessary step for building the data model, as we will see in the next subsection.

2. Develop Data Model.

By using the data subjects, attributes, and relationships identified in Appendices F through I, we developed a data model using the Semantic Object Modeling (SOM) technique in order to provide a better conceptual understanding of the requirements. This data model is shown in Appendix J.

The SOM technique is similar to the Entity-Relationship (E-R) technique in that both of these techniques facilitate the understanding and documenting of the user's data. The principal difference between these modeling tools is that the E-R model considers the entities and their relationships the atoms of the data model whereas the SOM takes the concept of semantic object as basic. These semantic objects are a map of the "things" that the users consider important. The semantic objects are the smallest distinguishable units the user may want to process. These objects may be decomposed later into smaller parts in the DBMS (or application), but during the modeling process, these smaller parts are of no interest to the users. (Kroenke, 1999, p. 102)

The IOTS data model consists of the following semantic objects:

- ITEM
- CUSTOMER
- INVENTORY
- INVOICE
- ORDERS
- TRANSACTION
- USERS
- WAREHOUSE

The semantic data model should be self-explanatory, but we assume that some of the audience may not be familiar with SOM technique. Therefore, we take the discussion one step further and comment on two of these semantic objects, CUSTOMER and ORDERS, that are shown in Figure 24.

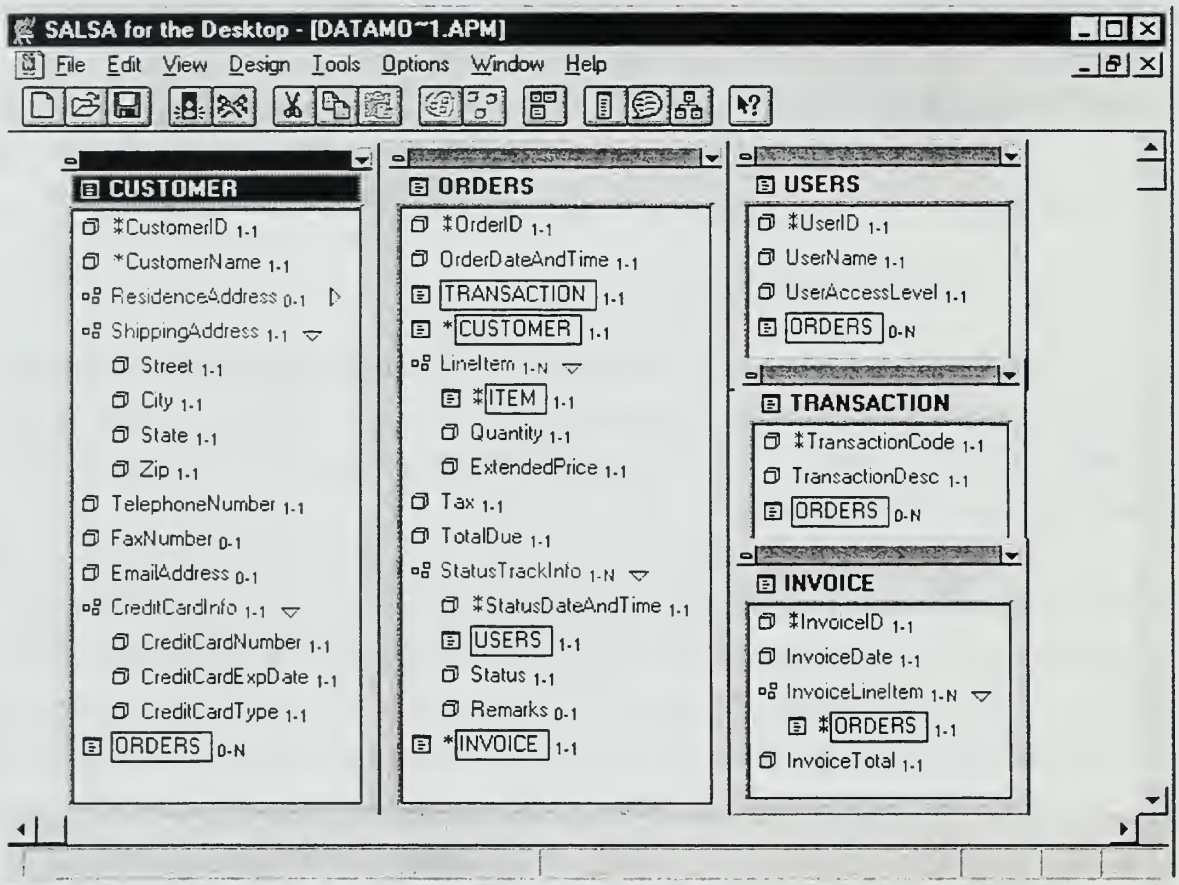


Figure 24. CUSTOMER and ORDERS Semantic Objects.

a. CUSTOMER

The CUSTOMER semantic object encapsulates the necessary information about customers. Each customer is given a unique identification number; the model shows the attribute "CustomerID" marked with double asterisks to indicate uniqueness. Customers also can be identified with their names, but more than one customer may have the same name. The model reflects this possibility by marking the "CustomerName" with

a single asterisk to indicate that duplicate names are allowed. The numbers on the right side of each attribute, object reference, or data group show minimum and maximum cardinalities. The model specifies "ResidenceAddress" as zero minimum cardinality whereas it specifies "ShippingAddress" as having a minimum cardinality of one; this means that the resident address is not required (optional) or can be null, but the shipping address must be provided for each customer. The "TelephoneNumber" takes one as maximum cardinality, which means that no more than one telephone number is allowed. Whereas the fax number can be null (not provided), the telephone number must be provided. Object references show relationships among semantic objects. The object reference, "ORDERS", has the cardinalities (0,N) which indicates that a customer may place many orders but may not have placed any orders at all.

b. ORDERS

This semantic object encapsulates the necessary information about orders. Each order has a unique identification number that will be used mainly for tracking purposes. Figure 25 shows a sample order.

Quick Supply Incorporation(QSI)
Customer Order

Order ID: FC-345-AB — Order Date: 01/18/2000
Customer Name: Alice Maher — Order Time: 8:00 AM

Item Number	Description	Quantity	Unit Price	Extended Price
1239567-34	Sanyodal TV 24x24	1	$420.00	$420.00
2345568-28	Vedio Model234	1	$220.00	$220.00
	Tax			$ 44.80
			Total Due:	$684.80

Figure 25. Sample Customer's Order

The reference object CUSTOMER, with (1,1) cardinalities, indicates that each order must be associated with one and only one customer. The reference object INVOICE, with cardinalities (1,1), indicates that each order must appear on one and only once invoice. Since it is possible for an invoice to contain more than one order, the INVOICE reference object is marked by a single asterisk. The reference object TRANSACTION, with cardinalities (1,1), indicates that each order must have one and

only one transaction type. Each transaction type requires slightly different handling; for example an order placed directly by a customer would be handled in a different way from an order placed by an LSM in terms of payment. Customers must provide their credit card numbers, but LSMs do not have to pay at all if their warehouses belong to the QSI. The reference object, ITEM, is embedded in a group item called "LineItem" as follows:

LineItem 1-N
1-1 *ITEM*
Quantity 1-1
ExtendedPrice 1-1

The ITEM reference is marked by double asterisks to indicate that the same item may not appear on the same order more than once. This reference object is also specified as (1,1) cardinality to show that each "LineItem" must have one and only one ITEM associated with it. Moreover, the reference object was grouped in a group data item, "LineItem" with (1,N) cardinality to indicate that a single order must have at least one "LineItem" and can have many "LineItems" as shown in Figure 25. The "StatusTrackInfo" keeps track of each status an order takes, as well as the time and date on which the status was assigned. It may also contain some explanatory remarks to inform the user why an order may have been deleted or canceled. For example, Figure 26 shows one possible sequence of the states assigned to the order presented in Figure 25.

Quick Supply Incorporation(QSI)
Tracking of Customer Order

Order ID: FC-345-AB Customer Name: Alice Maher

Status	Status Date	Status Time	Remarks
Waiting	01/18/2000	8:00AM	
Booked	01/18/2000	4:00PM	
Shipped	01/19/2000	9:00AM	
Delivered	01/20/2000	5:00PM	

Figure 26. Sample Track of a Customer's Order

The ORDERS data subject keeps track of the users who performed each transaction (state change) by recording the user's identification code along with every status change. Every status change must be associated with one and only one user. Figure

26 does not show the users who performed the status changes since the sample track in that figure is generated for the use of the customers. This kind of information is required for managers to track problems, and customers are not supposed to know the employees who took actions on their orders.

3. Transform the Data Model into Relations.

In this subsection, we followed the transformation methodology suggested by Kroenke, (Kroenke, 1999, pp. 163-185), to transform the semantic objects into relations. A semantic object may be mapped into one or more relations.

Table	Attributes
ITEM	ItemID, ItemDescreption, PartNumber, PurchasePrice, UnitOfPurchase, SalePrice, UnitOfSale, TaxableItem
CUSTOMER	CustomerID, CustomerName, ResAddrStreet, ResAddrCity, ResAddrState, ResAddrZip, ShipAddrStreet, ShipAddrCity, ShipAddrState, ShipAddrZip, TelephoneNumber, FaxNumber, EmailAddress, CreditCardNumber, CreditCardExpDate, CreditCardType
INVENTORY	ItemID-FK, WarehouseID-FK, Balance, ReorderLevel, ReorderQty
INVOICE	InvoiceID, InvoiceDate, InvoiceTotal
INVOICE_DETAILS	InvoiceID-FK, OrderID-FK
ORDERS	OrderID, OrderDateAndTime, TransactonCode-FK, Tax, TotalDue, CustomerID_FK, InvoiceID-FK
ORDERS_DETAILS	OrderID-FK, ItemID-FK, Quantity, UnitPrice, ExtendedPrice
ORDERS_STATUS	OrderID-FK, StatusDateAndTime, UserID-FK, Status, Remarks
TRANSACTION	TransactionCode, TransactionDesc
USERS	UserID, UserName, UserAccessLevel
WAREHOUSE	WarehouseID, WarehouseName, WareAddrStreet, WareAddrCity, WareAddrState, WareAddrZip, FaxNumber, TelephoneNumber

Table 4. The IOTS Relations

The following conventions are used: underlined attributes are primary keys or part of primary keys, foreign keys are extended with the postfix "-FK", and relation names are capitalized. If a foreign key is used locally it is also underlined. There are three major steps in the transformation process, which are *transformation*, *normalization*, and *table definition*.

a. Transformation

The transformation process results in the relations shown in Table 4.

b. Normalization

The data model is subject to a refinement process that modifies, adds, or deletes entities or their attributes in order to reflect the current business environment. As the refinement process occurs, the data model should be continually normalized. The normalization is a process that breaks an entity into two or more to eliminate undesirable modification anomalies. There are two types of modification anomalies, which are *deletion* and *insertion* anomalies. A *deletion anomaly* exists when facts are lost about two logical entities if one single deletion is performed. An *insertion anomaly* occurs when data cannot be inserted in a logical entity. Relational theorists chipped away at the types of modification anomalies to which relations are vulnerable and defined five normal forms which classify relations and provide techniques to prevent anomalies from arising. For our purposes, we validated the above relations through the first three normal forms. All the relations satisfy the first normal form because they meet the basic definition of a relation. All nonkey attributes are dependent on all of the keys. Therefore, the data model meets the criteria for the second normal form. Lastly, the data model also satisfies the third normal form by having no transitive dependencies.

There are still some possible improvements that can be made; here we mention one possible enhancement. The CUSTOMER table can be broken down into two or more tables. It currently contains the residential address, fax number, and email address which are all optional attributes; in most of the cases, the customer may not provide this information, which means space will be wasted if these fields are populated by null data. The same reasoning holds for fax numbers and email addresses. So a possible solution is to break down the CUSTOMER table into the following four tables:

- **CUSTOMER** (CustomerID, CustomerName, ShipAddrCity, ShipAddrState, ShipAddrZip, TelephoneNumber, CreditCardNumber, CreditCardExpDate, CreditCardType)

- **CUSTOMER_RESADDR**(CustomerID-FK, ResAddrStreet, ResAddrCity, ResAddrState, ResAddrZip)

- **CUSTOMER_FAX** (CustomerID-FK, FaxNumber)

- **CUSTOMER_EMAIL** (CustomerID-FK, EmailAddress)

However, there are costs associated with saving space. Breaking down the CUSTOMER table into these four new tables will complicate application development, and database administration and maintenance. Therefore, to keep the system simple, we have decided to keep the CUSTOMER relation unbroken.

c. Table Definition

Appendix K defines the attributes, primary keys, and foreign keys for all of the tables generated from the transformation process. This appendix will be the major input to the physical database design to be discussed in the next section. Special attention was paid to the selection of the primary keys. They were mainly selected from the attributes listed in Appendix H. Sometimes, there was a need to add new attributes to this list in order to uniquely identify some relations. Selecting primary keys must be carefully planned since they will be used to access relations for manipulation, modification, and retrieval purposes. These primary keys are required to facilitate the generation of the queries and reports listed in Appendices C and D. Moreover, the selection of the candidate attributes and primary keys should consider the available data attributes in the legacy data files in order to facilitate the future data migration to the new system.

C. DATABASE IMPLEMENTATION

In this section, we implement the relational database design for the IOTS. We select the DBMS product, define the IOTS database structure, and lay out the IOTS database system structure.

1. Selecting the DBMS Product

Oracle8i was chosen as a target DBMS for building the IOTS database structure. Whereas building the IOTS data model, in sections A and B, was independent of any DBMS product, implementing the IOTS database structure must evaluate the DBMS product being used since not all vendors commit to the same standards and conventions. We do not claim that Oracle8i is the best DBMS product in the market and the QSI needs to consider other DBMS products and decide on the best one that fits its needs. Usually making such decisions should be based on a special cost-benefit analysis that takes many factors into consideration, such as cost, legacy database migration, expected load, other alternatives, vendor's support, expected gains/loses, etc. This kind of study is outside the scope of this research. Oracle8i and its products represent an industrial-strength modern DBMS, and the following Oracle8i's strengths are very relevant to the QSI's objectives and requirements:

- Oracle8i adopts the Internet computing model where the QSI looks to get the benefit of the World Wide Web. Oracle8i also supports Java, CORBA, and other object-oriented languages.

- Oracle8i provides the security mechanisms required for the QSI to work in an open environment.

- Oracle8i has a suite of administrative tools that help in providing database backup and recovery, database migration, and network administration. Powerful tools are also available to help in application development and deployment.

- Oracle8i runs on different computer systems, which allows many choices of technology to the decision-makers in the QSI.

- Oracle8i supports all the following computing models: Host-based, Client/Server, Distributed Processing, and Web-enabled Computing, which gives the QSI the flexibility to switch from one computing model to another as needed, or to use more than one computing model at the same time.

- Oracle8i supports the Object-Relational model. The QSI is adopting to the object technology by using Java and CORBA; choosing Oracle8i will push

this object-oriented trend further and help the QSI define objects in their IOTS database.

2. Defining the IOTS Database Structure to the DBMS

Oracle8i provides two principal ways, as shown in Figure 27, for describing the structure of the database to the DBMS. The first way is to construct a text file that describes the structure of the database via the Data Definition Language (DDL). The text file names the tables, defines the indexes, and describes constraints and security restrictions. The latter way is to use the Schema Manager tool provided by ORACLE. This tool uses a graphical means for defining the structure of the database. With the Schema Manager (SM), an authorized user can view the database in a way similar to how the Windows 95 Explorer views the file structure. The SM makes it easy to add, modify, and delete tables, indexes, columns, views, primary keys, and foreign keys.

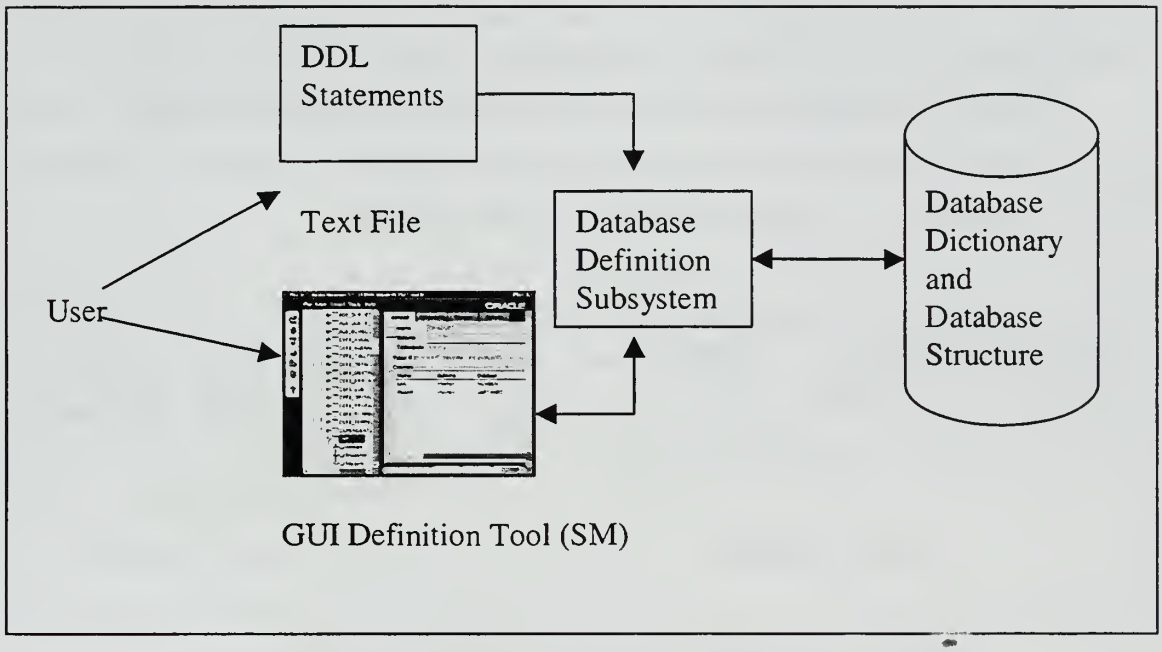


Figure 27. Database Structure Definition Process

In this research, we use the classical DDL approach. The data structure is defined in a text file as shown in Appendix L. This Appendix defines the tables, indexes, attributes, primary keys, foreign keys, and constraints of the IOTS database.

3. IOTS Database System Architecture

There are different system architectures that are used for multi-user database processing, such as teleprocessing, client/server, file sharing, distributed, and network database systems. The IOTS is intended to work in a client/server environment. Unlike teleprocessing, which involves a single computer, the client/server computing involves multiple computers. Some of these computers process the application program and are designated as clients. Another computer processes the database and is designated as a server. By adopting the client/server model, we have decided that the IOTS database will be unified rather than distributed. Distributed database processing allows partitioning and replication. Adopting the distributed database approach will increase parallelism, independence, flexibility, and availability, but at the same time this approach means greater expense, complexity, difficulty of control, and risk to security (Kroenke, 1999, pp. 280 –284). The choice between unified and distributed database processing was based on the nature of the QSI's business. The QSI seeks cost reduction, desires a system that can be managed easily, and highly values the confidentiality of its customers' records.

In the client/server database system, the application programs are placed on the clients' side and the DBMS and the operating systems are placed on the server. Figure 28 shows the programs involved in a client/server database system.

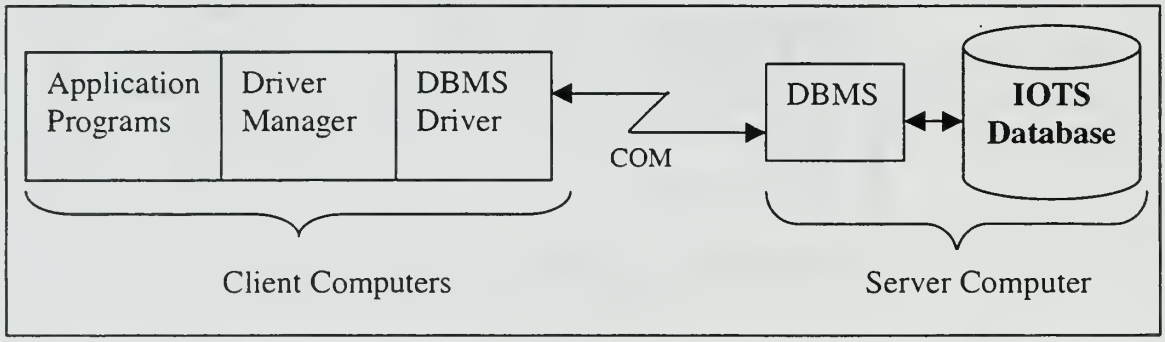


Figure 28. Programs in Client/Server Database Applications

The driver manager intermediates the application and the DBMS driver. When the application requests a connection with the database, the driver manager determines which DBMS driver is requested and loads it into memory. The function of the DBMS driver is to receive requests from the application, and format and deliver them to the DBMS. It also receives the responses from the DBMS and formats them to the application. A layer

of communication will handle the sent and received messages between the client and the server. In the next chapter, we will see that the ORB handles this communication layer. We will also see that the client application program, shown in the above figure, is actually the application server that will handle the actual clients' requests from the Web browsers. This application program will serve as the middle-tier between the clients and the database server.

D. SUMMARY

The IOTS database prototype was developed by using the requirements defined in Chapter II of this research. The development methodology advocated for the development of the data architecture is a three-step process. The steps of this process are: 1) define entities, attributes, and relationships, 2) develop the data model, and 3) transform the data model into a relational database schema. Oracle8i was used as a target DBMS for building the IOTS database structure. The IOTS database was designed to work in client/server multi-user data processing environment. In the next Chapter, we design and implement an application prototype that accesses the database already implemented.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DESIGN AND IMPLEMENTATION OF THE IOTS WEB BASED APPLICATION PROGRAM PROTOTYPE USING CORBA AND JDBC

In this chapter, we start by selecting an Object Request Broker (ORB) implementation from those available from different vendors. Next, we go through the application design process using the ORB that we selected. After that, we design and implement the prototype. Finally, we list and comment on some snapshots of the prototype.

A. ORB SELECTION

There are several ORB implementations available, each with different features. The ORB implementation that we used for the IOTS prototype was Visigenic Visibroker 3.4 that implements the specifications of CORBA 2.0. The main advantage of this product is that it has a stable IDL-to-Java mapping. Moreover, a client with access to the server via the Web is able to download the ORB on the fly. Although the selected ORB implementation is by Visigenic, the prototype was constructed using ORB independent services. Thus it is possible for the IOTS prototype to run on different ORB implementations assuming these implementations are CORBA compliant.

B. APPLICATION DESIGN PROCESS WITH VISIBROKER 3.4

As shown in Figure 29, the design process must adhere to the following steps in order to implement CORBA servers and clients. We illustrate this development process by creating a very simple client/server application. Understanding this example will help in understanding the more sophisticated system prototype that we build later in this chapter.

1. Perform Object Analysis and Design

First, we need to identify the objects that provide the system functionalities. In this Client/Server sample, the client will query the server about its local time. The server will be listening for the client's incoming requests and respond to the client with a message that contains the local time on the machine that the server is running on. In this example, the required service will be provided by a single object that we call TimeServer. This object will provide one single method that we call getTime(). This method will

return to the client with a message that contains the local time at the server. For example, when the client invokes this method on the TimeServer object, it will get a response that is similar to this message: "The time at the server is: 10:20:35".

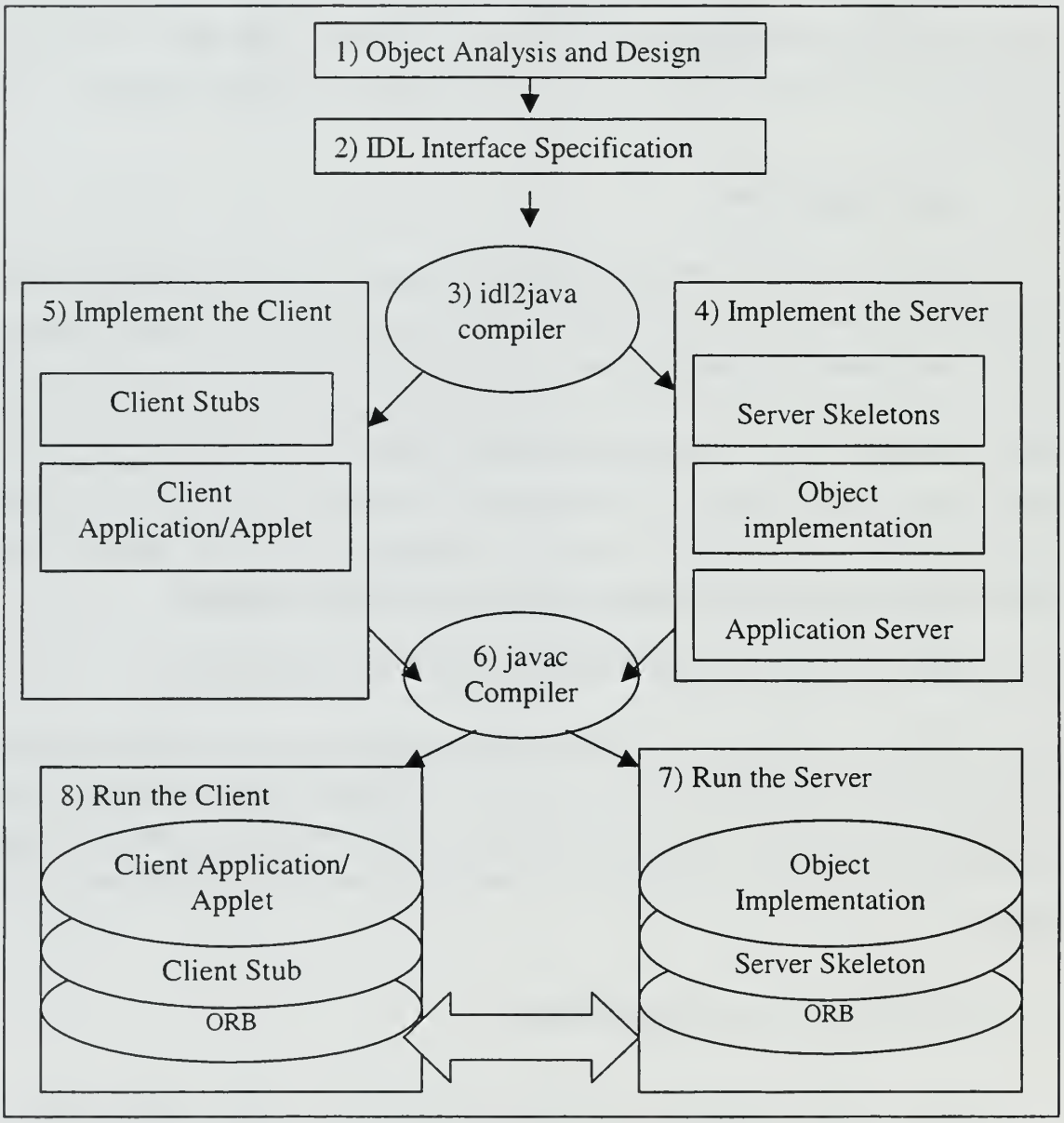


Figure 29. The Steps of Client/Server Development

2. Write the IDL Interface Specification for the Identified Objects

As we discussed in Chapter III, the IDL interface is a contract that defines the services provided by an object to its potential clients. Clients are not concerned about how this interface is actually implemented. The following is the IDL file for the TimeServer object. This file is saved as Example.idl.

```
module Example {
interface TimeServer {
string getTime();
};
};
```

2. Generate the Client Stubs and the Server Skeletons

In our prototype we used the Visibroker idl2java compiler for generating both of the client stubs and the server skeletons. If, for example, you wanted to implement the client in Java language and the server in C++, you would need to run both idl2java and idl2C++ compilers. Next, take the stubs generated by the idl2java compiler and the skeletons generated by the idl2C++ compiler and throw out the rest. In our example, we intend to implement both the client and the server with Java. So, we enter the following command:

```
prompt> idl2java Example.idl -no_comments -no_tie
```

The -no_comment parameter instructs the compiler not to generate comments in the generated files. The -no_tie indicates that we do not wish to use the tie mechanism in the server implementation. The compiler will generate the following files:

- **TimeServer.java**: This is the Java interface for the TimeServer object. We must provide code that implements this interface. This interface extends the root object in the ORB implementation. The following is the code of this interface:

```
package Example;
```

```
public interface TimeServer extends
com.inprise.vbroker.CORBA.Object {
public java.lang.String getTime();
}
```

- **TimeServerHelper.java**: This Java class provides helper services to the TimeServer clients. For example, this class enables the clients to use the narrow function that casts CORBA object references into a TimeServer type.

- **TimeServerHolder.java**: This class overcomes a Java limitation of passing parameters. Java natively only supports "in" parameters, but with the help of this class Java can pass "out" and "inout" parameters.

- **_st_TimeServer.java**: This class implements the client-side stub for the TimeServer object. It performs marshaling and unmarshaling of parameters.

- **_st_TimeServerImplBase.java**: The CORBA server-side skeleton for the TimeServer object is implemented in this class. One way to implement the TimeServer object is to extend this class.

- **_example_TimeServer.java**: The goal of this class is to facilitate the TimeServer implementation. It contains the constructors and empty definitions for the methods defined in the Example.idl file.

4. Add the Server Implementation Code

The developer must provide the implementation of the methods advertised in the interface. In other words, the business logic of the published IDL-interface must be provided. The following TimeServerImpl class shows how the TimeServer object is implemented by extending the skeleton class using the inheritance approach:

```
import java.util.Date;
public class TimeServerImpl extends Example._TimeServerImplBase {
public TimeServerImpl(java.lang.String name) {
 super(name); }
 public TimeServerImpl() {
```

```
 super(); }
 public java.lang.String getTime() {
  Date t = new Date();
  return ("The time at the server is: " + t.getHours()+":"
  + t.getMinutes() +":" + t.getSeconds());
 }
}
```

The following code acts as a driver. The main function initializes the ORB, instantiates the Basic Object Adapter (BOA), exports the TimeServerImpl object to the ORB, and waits for incoming requests.

```
//Server.java
public class Server {
public static void main(String[] args) {
try {
// Initialize the ORB
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
// Create the basic object adapter
org.omg.CORBA.BOA boa =
((com.visigenic.vbroker.orb.ORB) orb).BOA_init();
// Create the servant object
Example.TimeServer time =
new TimeServerImpl("localTime");
// Export the newly created object to the ORB
boa.obj_is_ready(time);
System.out.println(" Msg from server.. time: " +
time + " is ready");
// Ready to serve requests
boa.impl_is_ready();
}
```

```
catch (org.omg.CORBA.SystemException ex)
{
System.err.println("System exception in TimeServer");
System.err.println(ex);
}//catch
}//main
}//Server
```

5. Write the Client Implementation

Before the client can use the services provided by the server, it must obtain an object reference to that server. There are various ways for the client to obtain this reference. In the following client implementation, we use the bind() method to obtain a reference to the TimeServer object, whereas in the IOTS prototype the CORBA naming service is used. The bind method is ORB-dependent, whereas the naming service is not. Therefore, applications that utilize the naming service are much more portable among various ORB implementations.

```
// Client.java
class Client {
public static void main (String [] args) {
try {
// Initialize the ORB
System.out.println("Initializing the orb");
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);
// Bind to the TimeServer object
System.out.println("binding to the TimeServer Object");
Example.TimeServer time = Example.TimeServerHelper.bind(orb,
"localTime");
// Get the time
System.out.println(time.getTime());
```

```
} catch (org.omg.CORBA.SystemException ex) {
System.err.println("System exception in Client");
System.err.println(ex);
}//catch
}//main
}//Client
```

6. Compile the Client, the Server, and the Server Implementation Classes

In order for the compilation to be successful, the compiler must have access to the files generated in step three above. We use the JDK1.2.2 compiler and enter the following commands:

```
prompt> javac TimeServerImpl.java
prompt> javac Server.java
prompt> javac Client.java
```

7. Run the Server

Before running the server, the Visibroker requires that the OSAgent service be running. The OSAgent provides location service to help objects that are using the bind() operation locate each other. The following two commands start the OSAgent and the Server:

```
prompt> start/min OSagent -c
prompt> java Server
```

8. Start the Client

The client is started by using the following command

```
prompt> java Client
```

C. DESIGN AND IMPLEMENTATION OF THE IOTS PROTOTYPE

In this section, we first discuss the QSI's business model, build the system architecture, design the system prototype, and finally implement the prototype.

1. Business Model

Unless an accurate business model is built for the QSI, the results will be "garbage in, garbage out." Figure 30 lays out the overall picture of this model. The QSI holds stocks of different kinds in its main store. The main store forms a buffer between suppliers and customers. These stocks are replenished by deliveries from suppliers and they are reduced to meet demands from customers. In Figure 1, the local warehouses are considered the only customers of the QSI. Here, Internet customers are added. This new category of customers must be accommodated in order to meet the requirements of the system discussed in Chapter II. The inventory is held in stock in the main store to allow for operations to continue smoothly in spite of the uncertainty in supply and demand. The QSI struggles to make the stock holdings as efficient as possible. The QSI always needs to know what items to stock, when to place orders, and what quantity should be ordered. So, in addition to the IOTS, the QSI is running an Inventory Control and Management System (ICMS). There is no clear boundary between these two systems since in most of the cases they are interrelated. For example, the ICMS calculates the amount of stock that should be held in the main store, using the amounts ordered through the IOTS.

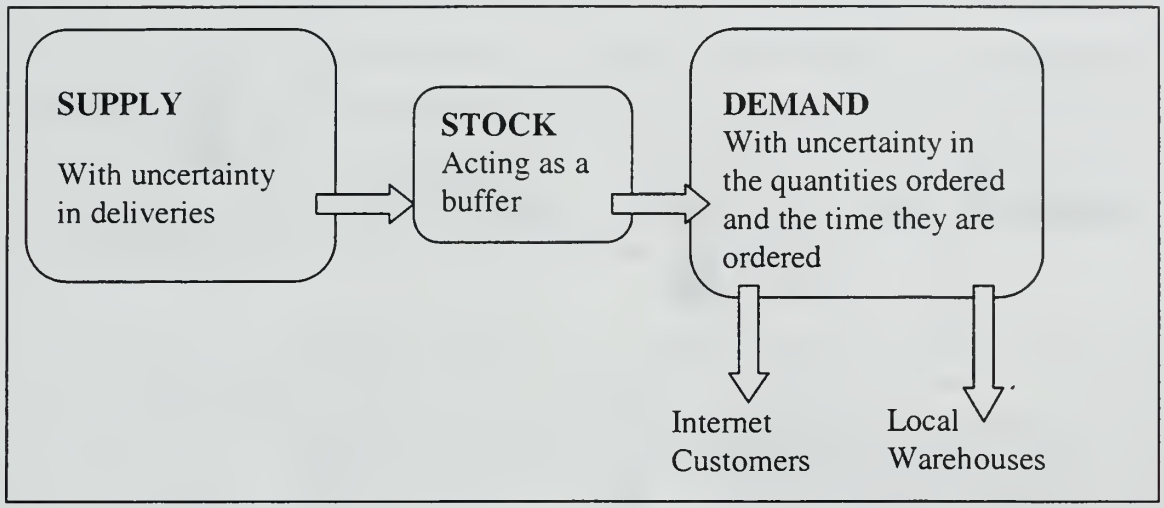


Figure 30. The QSI's Business Model

2. System Architecture

The system architecture defines the basic system components and the relationships among them. It depends on the nature of the business and the requirements that must be achieved. Figure 31 shows the IOTS basic architecture. The Figure shows different applications for different users. In traditional businesses, buyers interact with their sellers directly, and both interact with the operations and support groups. On the Internet, the main means of interaction among these parties is via applications that place their retained states in the IOTS centralized database. The three major components that make up the system's architecture are

- Customer Components
- Local Warehouse Components
- Seller Components and Servers

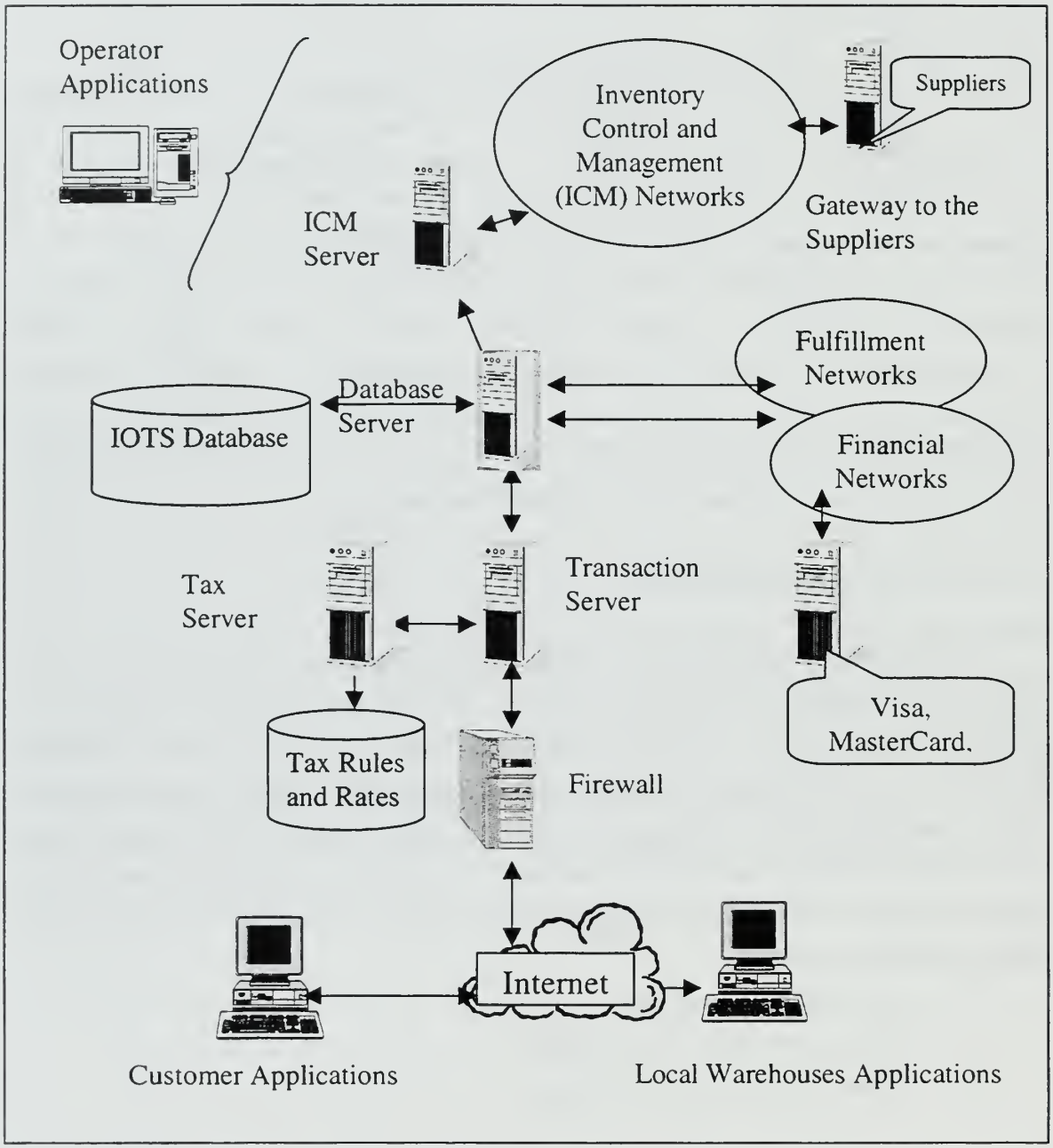


Figure 31. System Architecture

a. Customer Components

The primary tool used by customers is the World Wide Web. The IOTS will be restricted to the browsers' capabilities. We will see later how Java/CORBA can extend these capabilities to benefit the customer and to overcome some of the web browsers' limitations. The customers' applications will provide the following functionalites:

- Customers will be able to log on to the system, browse the available products, populate an order form, and submit that order. The order form will provide the customer with the means of editing personal information, such as shipping address, billing address, and payment method. Moreover, the order form will be easy to use and enable the user to adjust the item quantities.

- The system will make it easy for customers to register and track their orders.

- Walk-in customers who do not want to register will also be served, but they will have to enter all the required information, i.e. name, address, etc. every time they decide to buy. Order tracking service will be provided for registered customers only.

b. Local Warehouse Components

This category provides services to the Local Store Managers (LSMs) who will be using the Web browsers. We make this category separate from the ordinary customers since the LSMs will be given more authority. The IOTS is expected to provide them with additional functionalities; therefore, they need special client applications and a distinguished user interface that meets their needs. The following three services are some examples of the many services that will be provided to LSMs:

- LSMs will be able to track orders they place, either one by one, or by viewing the final status on all orders placed within a date range.

- LSMs will be able to view the inventory levels of their stores. Note that the inventory levels of all the local stores are stored in the IOTS centralized database. Therefore, local warehouses do not record their inventory levels locally.

- LSMs will be able to order reports from the CMO and be able to track these reports.

c. Seller Components and Servers

The seller or the merchant provides the core functionalities of the IOTS by running and maintaining the following components:

- **Transaction Server**. The transaction server will keep track of all the transactions placed by the Internet customers and the LSMs. It will record what was ordered, how much was ordered, and who ordered it. This server will be running the IOTS application prototype that will provide the transaction service in the middle tier. More details about the application prototype will follow later in this chapter.

- **Database Server**. This server will be running the DBMS and will enable the users to access the IOTS database. This database was designed and implemented in chapter III using ORACLE8i.

- **Fulfillment Networks**. This component will take care of packing and shipping orders. The seller has access to the transaction statements and can mark orders as fulfilled and send the appropriate notification to the customer.

- **Financial Networks and Payment Gateway**. This group will take care of all the payment instruments in the system. For example, the sellers will connect the credit card agencies through the payment gateway in order to authorize the transaction and later to settle the payment.

- **Inventory Control and Management (ICM) Network**. This part of the QSI is responsible for managing the inventory levels across all the warehouses. The ICM group decides what items should be stocked, when an order from the suppliers should be placed, and how much should be ordered. Inventory levels are controlled through a set of applications that run on the ICM Server. The connection to the suppliers is performed through the suppliers' Gateway Server.

- **Tax Server**. In the United States, the buyer owes taxes to the state governments, counties, and localities (cities). There are over 6,000 rules about sales tax and they change frequently (Treese, 1998, p. 296). Figure 32 shows the required inputs for the Tax server to perform tax calculations. For the sake of simplicity, the IOTS prototype applies a fixed rate of seven percent on taxable items. However, in practice, the tax must be calculated in the right way; otherwise it will not be legal. Placing the Tax server on a stand-alone machine will make it easy to manage and maintain without affecting the other parts of the system.

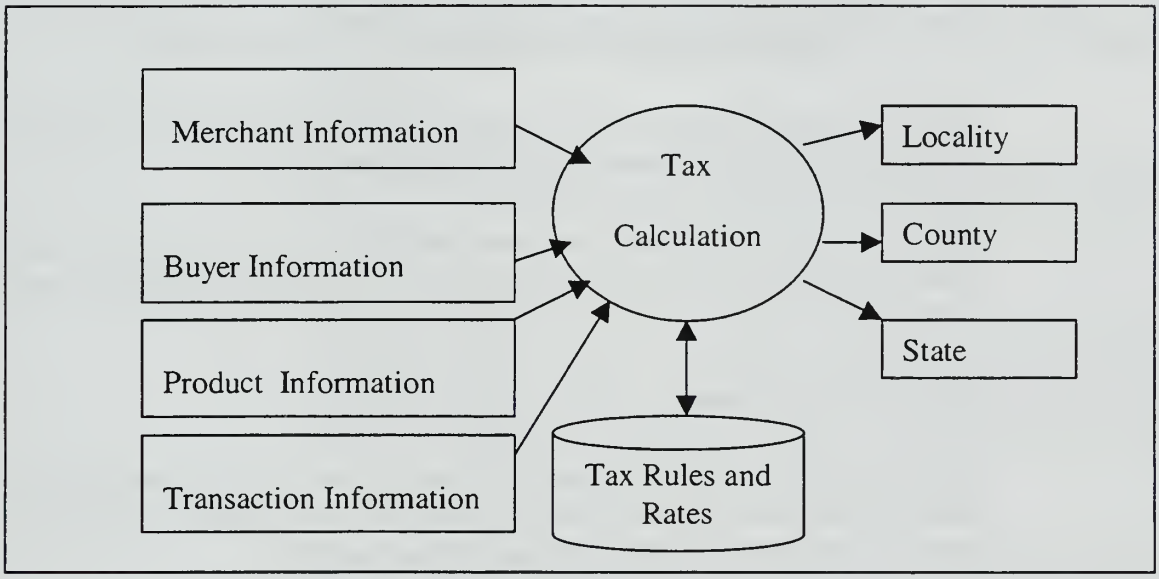


Figure 32. U.S Sales Tax Calculation (Treese, 1998, p. 296)

- **Operator Applications**. This set of applications is used for configuring, maintaining, administering, and monitoring all the components that exists under the seller's control.

3. Prototype Design

There is no clear cut wisdom on how to conduct the design process of a distributed system. Most of the design methodologies that we surveyed agree on two major steps. The first step is to start with the problem statements and divide the required functionalities among objects. The latter step is to define the services provided by each

single object and the interactions among them. This will ensure that the system satisfies the requirements and provides the expected services.

In the design of the IOTS prototype, we use a "quick-and-dirty" procedure that goes side by side with the implementation in order to get early feedback on whether our design is on the right track or not. This works fine for a prototype which only covers a small set of the overall IOTS functionalities. In order to provide a complete design of the IOTS, however, a more formal design process must be followed; otherwise, with hundreds of objects it can quickly become complicated. Designing distributed applications is quite complex and even with the simple prototype we could not keep it consistent, structured, simple, and straightforward without performing these steps:

- Define the user interfaces of the system to ensure that all of the IOTS's required services will be provided to the authorized users.
- Pick one of these interfaces and generate different use cases. For more details on use-case analysis, the audience may consult with (Booch, 1994) or with (Blaha, 1998).
- Define the object model.
- Break down each single use case into one or more scenarios and use the sequence diagrams to illustrate how objects that were defined in the object model will interact in order to implement each scenario.
- Write the Interface Definition Language (IDL) specifications.

a. User Interfaces

The IOTS is broken down into these various interfaces in order to ensure that it provides the functionalities that satisfies the requirements listed in Chapter II of this research:

- **Customers Interface**. Used by customers to place and track orders over the Internet. It also provides the customers with the necessary services, i.e. enabling them to modify shipping and billing addresses and giving them directions on how to use the interface and how to contact the sellers.
- **LSMs Interface**. Used by LSMs to place and track orders for their warehouses. It also enables them to order and track reports from

the CMO. Appendix D shows a list of these reports. Moreover, using this interface, the LSMs will be able to perform the queries listed in Appendix C.

- **Purchase Interface**. This interface is used by the GSMs to place and track orders from the suppliers.

- **Inventory Control and Management (ICM) Interface**. The GSMs will use the ICM interface to control the inventory levels across all the warehouses. This interface will help the GSMs decide what items should be placed in stock, what items should be removed, when to place an order, and how much to order. This interface will also be used to generate the various reports listed in Appendix D.

- **Fulfillment Interface**. Will be used by fulfillment group to help in packing the booked orders and to mark these orders as shipped when they go out.

- **Payment Interface**. This interface will help to authorize the transaction and to settle the payment later. Paid orders will be marked as "PAID." Paid orders will eventually be closed, removed from the database to save some space, and archived for later possible use. This interface will perform the billing.

- **Other Interfaces**. This includes operator interface and customer service interface. Operators use the operator interface to maintain, administer, and monitor the system. The customer interface is used to provide services to customers. For example, a customer may phone, inquiring about his order and a customer representative will use this interface to serve that customer.

b. Use Cases

Among the interfaces listed above, we chose to design and implement some of the functionalities provided by the LSMs Interface. In practice, the use cases are generated from the interactions with the end users. For our purposes, we analyze the roles of the LSMs and the requirements defined in Chapter II, and generate the following set of use cases:

- An LSM places an order.

- An LSM finds out about the status of an order.
- An LSM finds out about the status of a group of orders within a date range.
- An LSM adds an item to an existing order.
- An LSM inquires about the available balance of a given inventory.
- An LSM creates, adds, and modifies a customer record.
- An LSM modifies the information about his or her warehouse, like an address, telephone number, etc.
- An LSM lists all the transactions made on a given inventory.
- An LSM orders a report.
- An LSM finds out about the status of an ordered report.

c. Object Model

After the use cases are defined, the object model is developed. The basic objects in the system are defined and the interaction among these objects determined. The Unified Modeling Language (UML) is used to facilitate the design process. Figure 33 shows a simplified object model of the IOTS prototype.

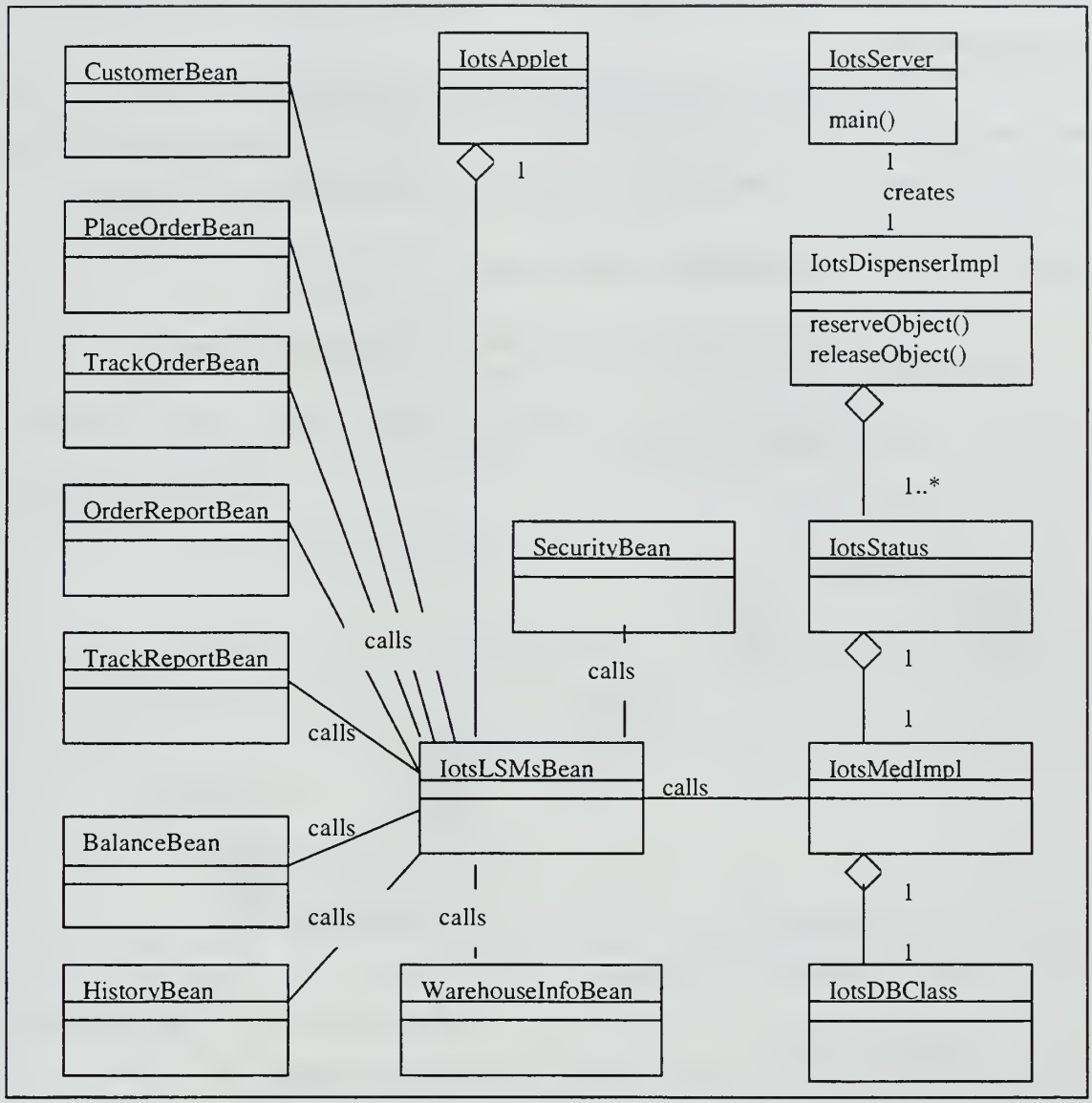


Figure 33. The IOTS Object Model

d. Scenarios and Sequence Diagrams

The sequence diagrams illustrate how objects will interact in a specific scenario. Sequence diagrams make the developer visualize how the distributed objects will interact to execute a given scenario. Moreover, sequence diagrams will help the developers discover any mismatches between the clients and servers objects early in the design process. Finding such discrepancies at the beginning of the design phase will

reduce the cost of fixing them and will reduce the possible bugs that may appear during the implementation.

Each use case can be broken down into one or more scenarios and each single scenario can be represented with a sequence diagram. For example, the following are some of scenarios that were defined for the "An LSM places an order" use case:

- An LSM places a valid order.
- An LSM places an order for an item that does not exist.
- An LSM places an order with quantity equal to zero or greater than the reorder-level.

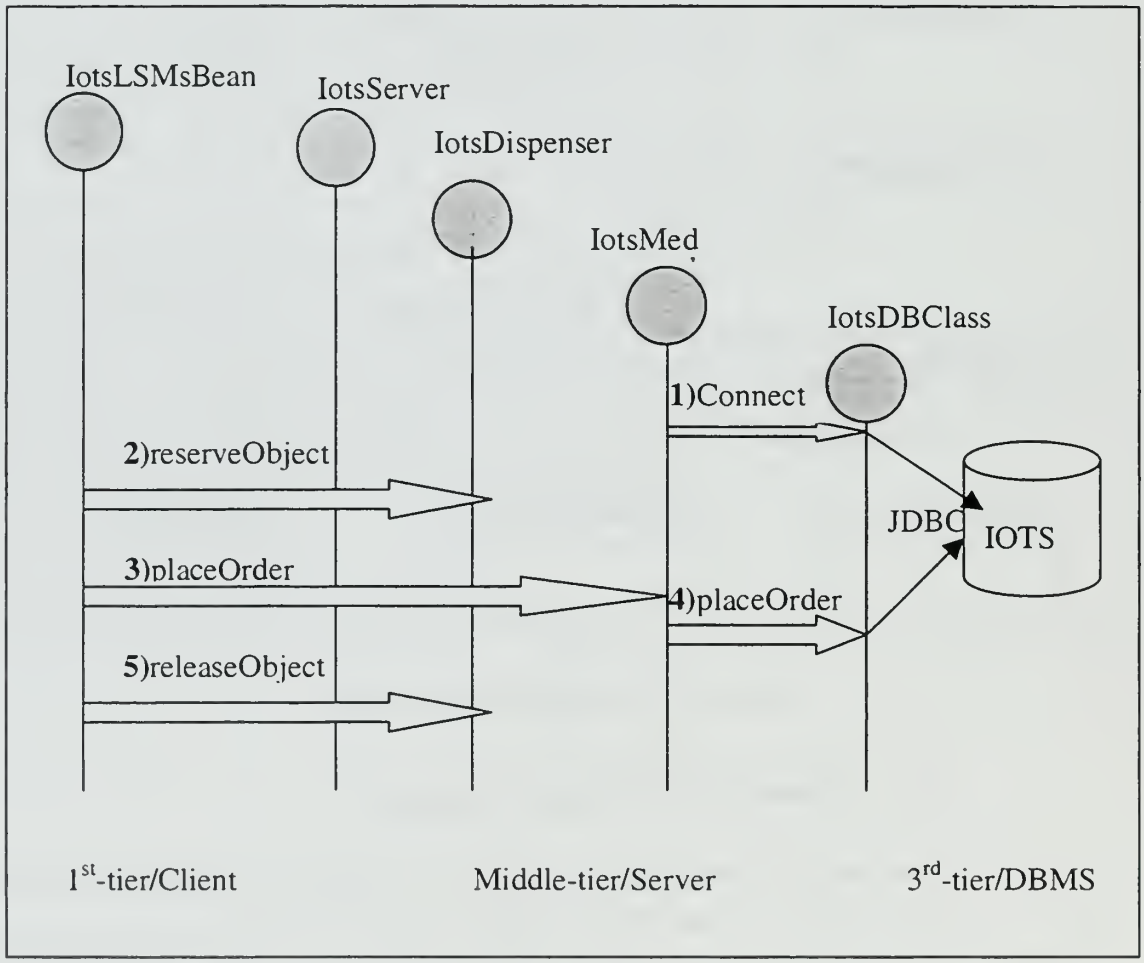


Figure 34. The Sequence Diagram for Placing a Valid Order

- An LSM places an order on behalf of the warehouse.
- An LSM places an order on behalf of a customer, and the LSM does not provide valid customer information.
- An LSM submits an order with no items.
- An LSM places an order for an item that has no available balance in the main store.
- An LSM places an order for an item that is no longer supplied.

Figure 34 shows the sequence diagram for "An LSM places a valid order" scenario as an example.

c. IDL Specifications

The last step of the design process is to write the formal IDL specifications that will serve as a contract between clients and servers objects. We have briefly discussed IDL in Chapter III. The complete IDL specification file for the IOTS prototype is listed in Appendix M.

4. Prototype Implementation

The IOTS prototype is implemented as a three-tiered client/server application as shown in Figure 35. In Chapter III, we discussed the advantages of three-tiered applications over two-tiered applications.

The first tier, the client, is the user interface for the LSMs, which will be running on the Web browsers. The second tier, the server, can serve both HTTP and CORBA. The server will encapsulate the business logic and interact with the client via CORBA/IIOP. CORBA objects will interact with each other using CORBA Object Request Broker (ORB), and they will talk to the third tier using SQL/JDBC. The third tier consists of the IOTS database that will be accessed by the CORBA objects. The IOTS database was designed and implemented in chapter IV of this research. CORBA/Java will be used to encapsulate most of the third-tier functions.

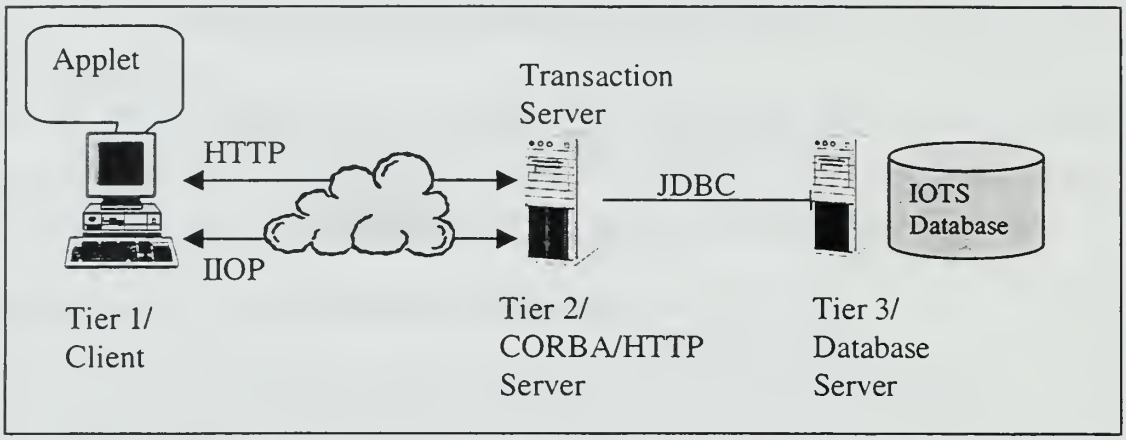


Figure 35. The IOTS in a Three-tiered Client/Server Architecture

As shown in Figure 36, the IOTS prototype consists of the following three categories of components:

- Client components
- Transaction Server components
- Database Server components.

The third category was discussed in Chapter IV so we limit the discussion to the first two categories. Appendix O lists all the software components used in the development of the IOTS prototype.

a. Client Components

A Web browser downloads the HTML page that includes references to the embedded Java applets using the HTTP. The browser also downloads the required ORB classes. The HTML was mainly used to embed the client's applet. In practice, the QSI may consider some other markup languages, such as the Extended Markup Language (XML), and the Standard Generalized Markup Language (SGML). For a comparison of these languages, the reader may refer to (Norman, 1998).

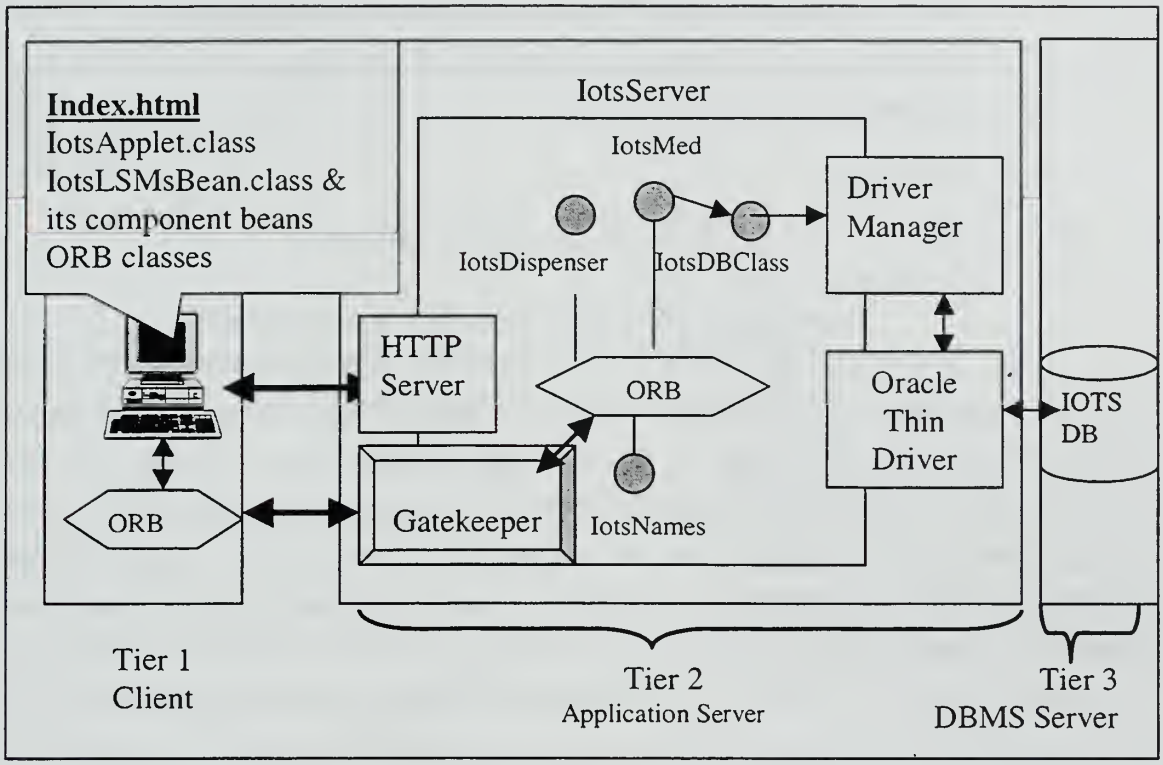


Figure 36. The IOTS Prototype Components

The following is the source HTML code for the "index.html" file that we wrote to work in the IOTS prototype. The numbers in braces are not part of the code, and they will be used later to explain the applet's parameters:

```
<HTML>
<h1>Quick Supply Inc. </h1>
<hr> <center>
<APPLET CODE=IotsApplet.class WIDTH =550 HEIGHT=350>
(1      <param name=org.omg.CORBA.ORBClass
         value=com.visigenic.vbroker.orb.ORB>
(2)     <param name=ORBgatekeeperIOR
        value=http://computerb:15000/gatekeeper.ior>
(3)      param name=ORBalwaysProxy
```

```
        value=true>
(4      <param name=ORBservices value=CosNaming>
<param name=SVCnameroot value=IotsNames>
</APPLET>
</center>
<hr>
</HTML>
```

The above code contains some parameters. The first parameter forces the applet to download the ORB classes from the Web server. The second parameter indicates the port and the computer on which the Gatekeeper is running. The third parameter forces the client to use the Gatekeeper before it can access the CORBA objects on the transaction server. This parameter is used to enforce security and to increase performance. With this parameter, the ORB will not try to connect directly to server objects even when the server objects are reachable by the client. Therefore, the ORB will not waste time trying to connect to some unreachable objects. More information on the Gatekeeper will be provided under server components. The last parameter indicates that the applet requires the CORBA naming service to find the server objects and that the root-naming context is "IotsNames".

Since our prototype was designed with JDK version1.2.2, the Web browsers that contain an older JVM will not be able to execute our prototype. So, clients need to download the JVM plug-in from Sun's Javasoft Web site. We have converted the above HTML code using the HTMLConverter from Sun. The HTMLConverter adds some tags to the code. These additional tags will enforce Web browsers to use the JVM plug-ins that you downloaded instead of the Web browser's baked-in JVM. If the required plug-ins were not up-to-date the client will be guided to the Web site where the required version can be downloaded. After that, we designed a Web page with Microsoft Front Page 98 and embedded the converted applet in it. The final HTML code is shown in Appendix N.

The *IotsApplet.class* references the *IotsLSMsBean.class* that will initialize the orb and return a reference to a server object. The connection between the client applet and the server object will be performed over IIOP and not over HTTP. This connection shows the beauty of the Object Web. Java is necessary, but not sufficient for creating the

Object Web. It needs to work side by side with CORBA to achieve this goal. The client of the IOTS prototype is packaged into a single HTML page (index.html) that contains several components. Interestingly, clients will be able to interact with server objects by clicking any of these components without switching page's context to obtain each response.

b. Transaction Server Components

The middle tier contains the following components:

- **HTTP Server**

As mentioned earlier, the HTTP Server will enable clients to download the HTML pages, the referenced Java classes, and the ORB classes. In the IOTS prototype, Microsoft Internet Information Server (IIS) was used to provide HTTP service. In practice, the QSI may need to consider performance, security, scalability, and other related issues before selecting the HTTP server.

- **Gatekeeper**

Web browsers impose the following restrictions on Java applets:

- Applets are only allowed to connect to the host from which they were downloaded.
- Applets are only allowed to accept connections from the host from which they were down loaded.

The Gatekeeper is used to provide a way that works with these restrictions. To get around the first restriction, the Gatekeeper acts like an IIOP firewall proxy that forwards any calls from the applet to the target object. The second restriction is also handled by the Gatekeeper. The ORB sets up a special connection between the applet and the gatekeeper. The Gatekeeper uses this connection to forward the callback from the server object to the client. (Inprise, 1999, p. 2-2)

- **IotsServer.class**

This class implements the main method for the server. First, it initializes the ORB and creates a new IotsDispenser object. Next, it registers this object with the ORB. After that, the newly-created IotsDispenserImpl object is registered with the naming service. Finally, the server waits for incoming requests. The source code for this class and all other Java classes is provided in Appendix P.

- **IotsDispenserImpl.class**

This class implements the IotsDispenser interface. It was reused from (Orfali, 1998, pp. 944-946) after we changed it slightly. It creates a pool of IotsMedImpl objects and stores their references in an array of IotsStatus objects. Each IotsMedImpl object is registered to the ORB.

This class uses its reserveObject method to reserve one of the objects from the object pool to the client. The releaseObject method is used to release the object from the client and return it to the pool. Objects are located to clients on demand. Each object runs in its own thread and maintains a live connection with the database. Therefore, once one of these objects is allocated to a client, the client does not have to wait for a connection to be established. This will increase the performance and will make it possible for the client to be served without much delay.

- **IotsStatus.class**

The IotsDispenserImpl will contain an array of these objects. IotsStatus contains two fields: the first is a reference to the server object and the second is a flag that indicates whether and object is reserved by a client or not. If an object is reserved to a client, other clients will not be able to access it, they will reserve another object from the pool, instead.

- **IotsMedImpl.class**

This class creates a new IotsDBClass object that is connected to the database in the third tier via JDBC. It implements all the client's required methods. Each one of these methods corresponds to a method implemented in the IotsDBClass object.

- **IotsDBClass.class**

This class encapsulates all the methods needed to interact with the database in order to provide the client with the required service.

- **Driver Manager and JDBC Thin Driver.**

The functionalities of these drivers were discussed in Chapter IV. The ORACLE thin driver was used to maintain a JDBC connection between the IotsDBClass object and the IOTS database server in the third tier.

D. SYSTEM OPERATION SCENARIO

The steps listed below should be followed so that the IOTS can provide services to clients:

- Install Java and Visibroker ORB on the machine that will host the CORBA server.
- Start the Visibroker OSAgent. This tool is used transparently by the naming service. In windows NT, the OSagent can be started automatically by adding it to the system services or by typing the following command:

  ```
  prompt> start/min osagent -c
  ```

- Start the CORBA naming service by entering the following command:

  ```
  prompt>  start /min vbj
  -DORBservices=CosNaming
  -DSVCnameroot=IotsNames
  -DJDKrenamingBub
  com.visigenic.vbroker.services.CosNaming.ExtFactory IotsNames
  %VBROKER_ADM%\cosnm_dir\IotsNames.log
  ```

- Start the Gatekeeper by entering the following command:

  ```
  prompt> start/min gatekeeper
  ```

- Start the server by entering the following command:

  ```
  prompt> start/min vbj
  ```

-DORBservices=CosNaming

-DSVCnameroot=IotsNames IotsServer <num_of_ con_objs>

The *num_of_con_objs* is the number of live connection objects that must be ready in the connection pool in order to serve customers. The default of this parameter is four.

- Install the HTML files and the Java applet classes on the Web server
- Clients can access the server by entering the appropriate URL in their browsers. For our IOTS prototype, the URL is http://computerb.otoom/thesis/index.htm.

The following are some snapshots obtained after running the IOTS prototype. Figure 37 shows the logon screen on which authorized LSMs are authenticated to the system.

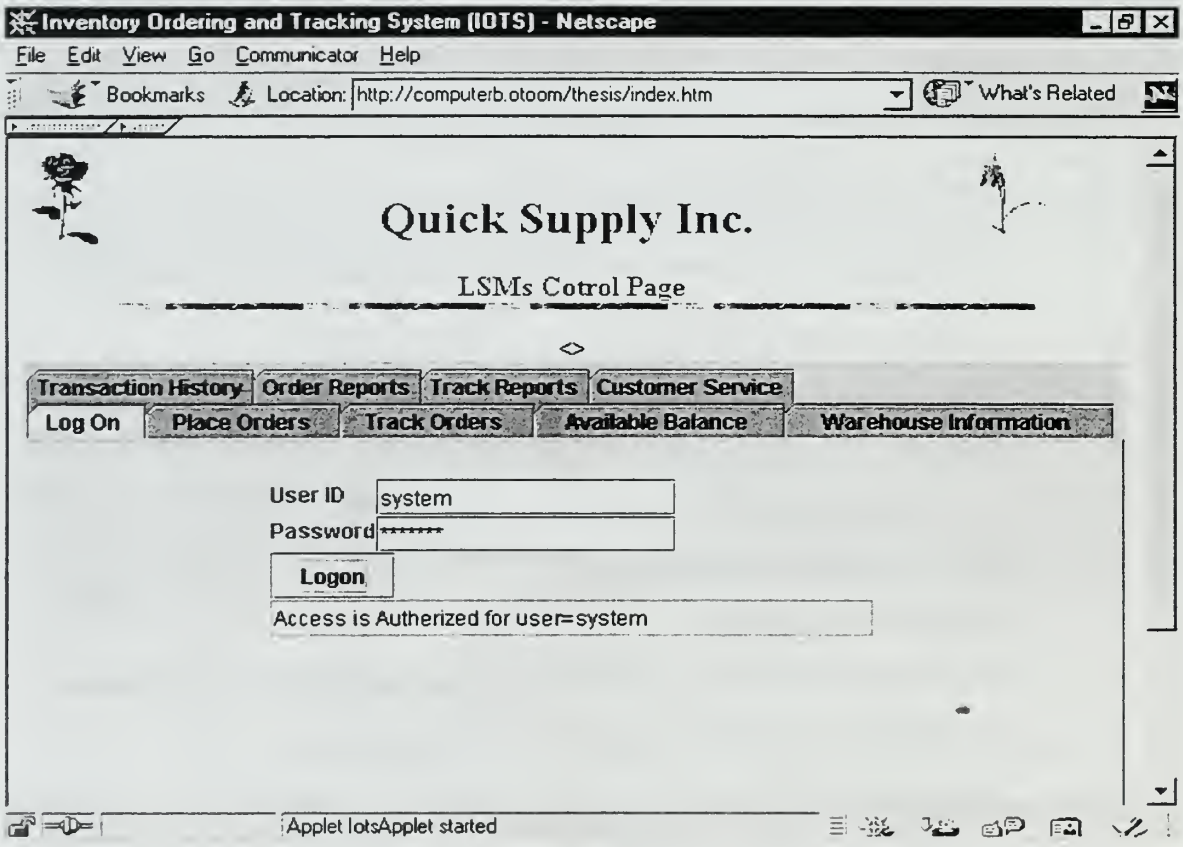


Figure 37. Logon Form

Because some of the information about customers, i.e. addresses, credit card types and expiration dates, electronic mail addresses, and phone and fax numbers change frequently, LSMs use the form shown in Figure 38 in order to keep customers' records up to date.

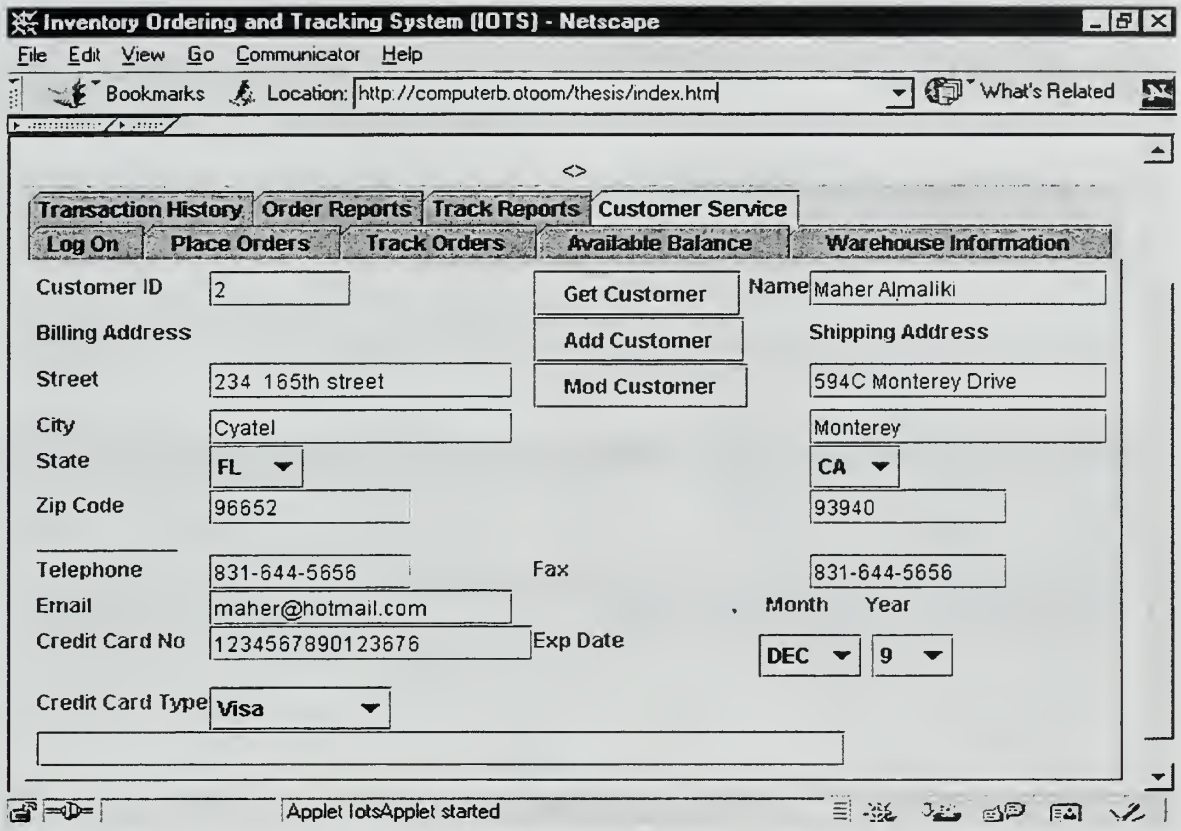


Figure 38. Customer Form

LSMs use the form shown in Figure 39 in order to keep their warehouses' identification records up-to-date.

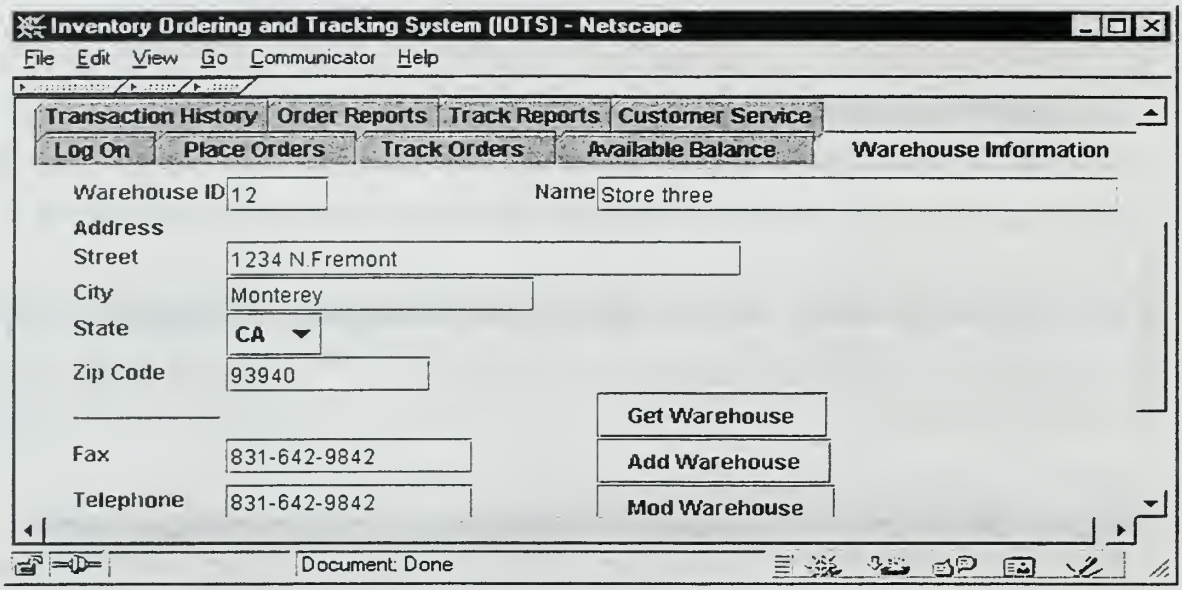


Figure 39. Warehouse Form

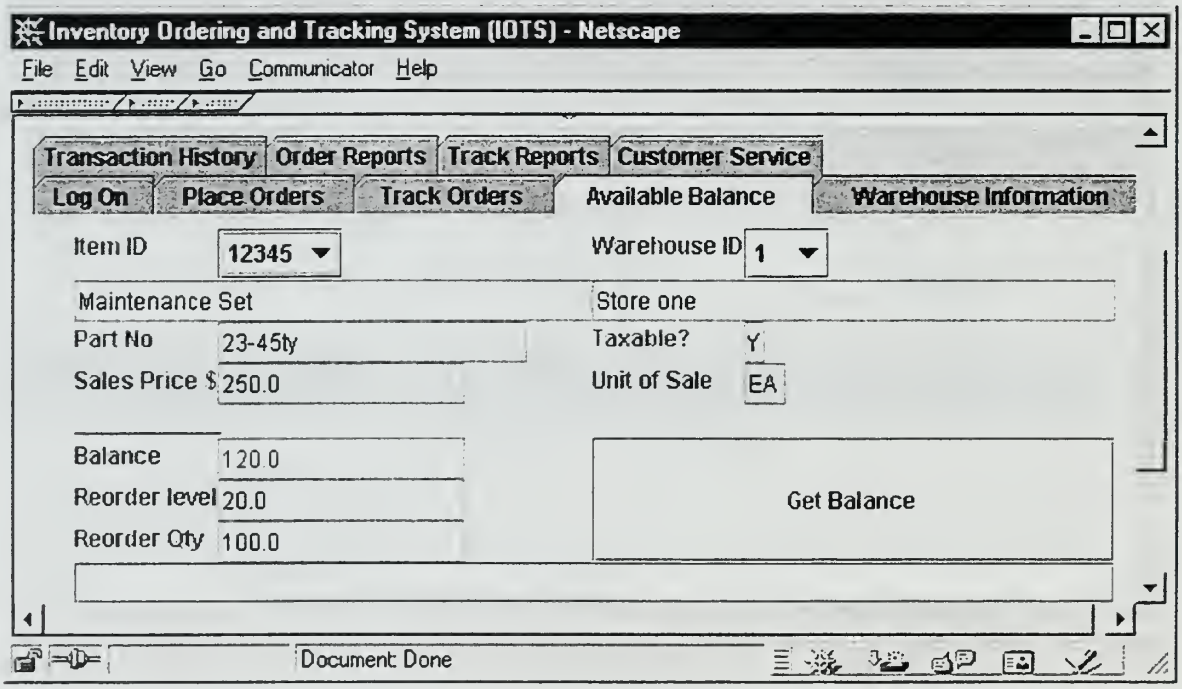


Figure 40. Inventory Levels Form

The form shown in Figure 40 enables LSMs to inquire about the inventory levels.

The Form shown in Figure 41 enables LSMs to place orders. An LSM can choose to place an order for a customer or to place an order for the local warehouses by choosing the appropriate transaction code. The form below shows an order placed by a LSM for a customer. The LSM is able to select from the items offered for sale, add to and delete from a shopping cart, adjust ordered quantities, and submit the order once ready. After submitting the order, the LSM will be provided with a unique number that can be used for tracking purposes. The LSM will pass this number onto the customer.

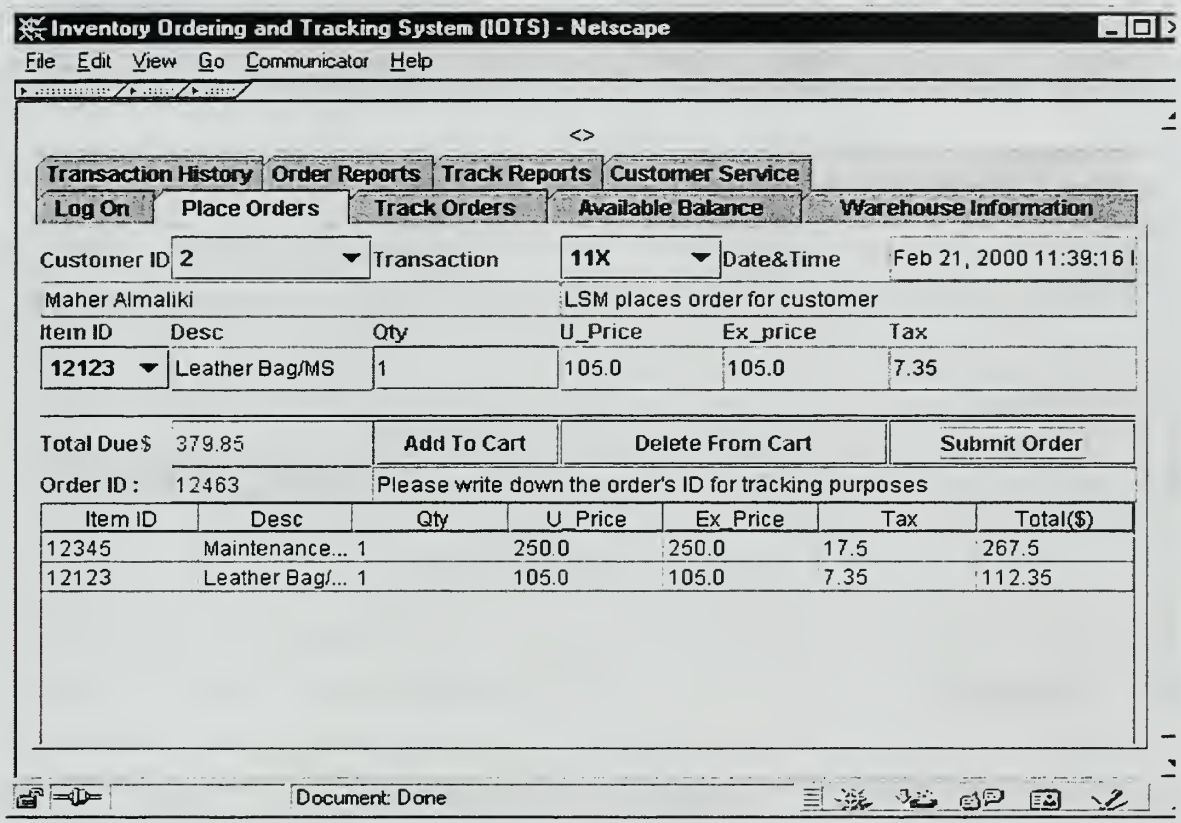


Figure 41. Place Orders Form

Finally, Figure 42 shows the form via which an LSM can track a single order by entering the exact order identification number, or a group of orders by entering the date range in which these orders occurred. The figure shows a list of the orders placed within a date range along with the final status on each order. The LSM can obtain a more detailed track by selecting the order identification number. A detailed track for the order number "12459" is shown as an example.

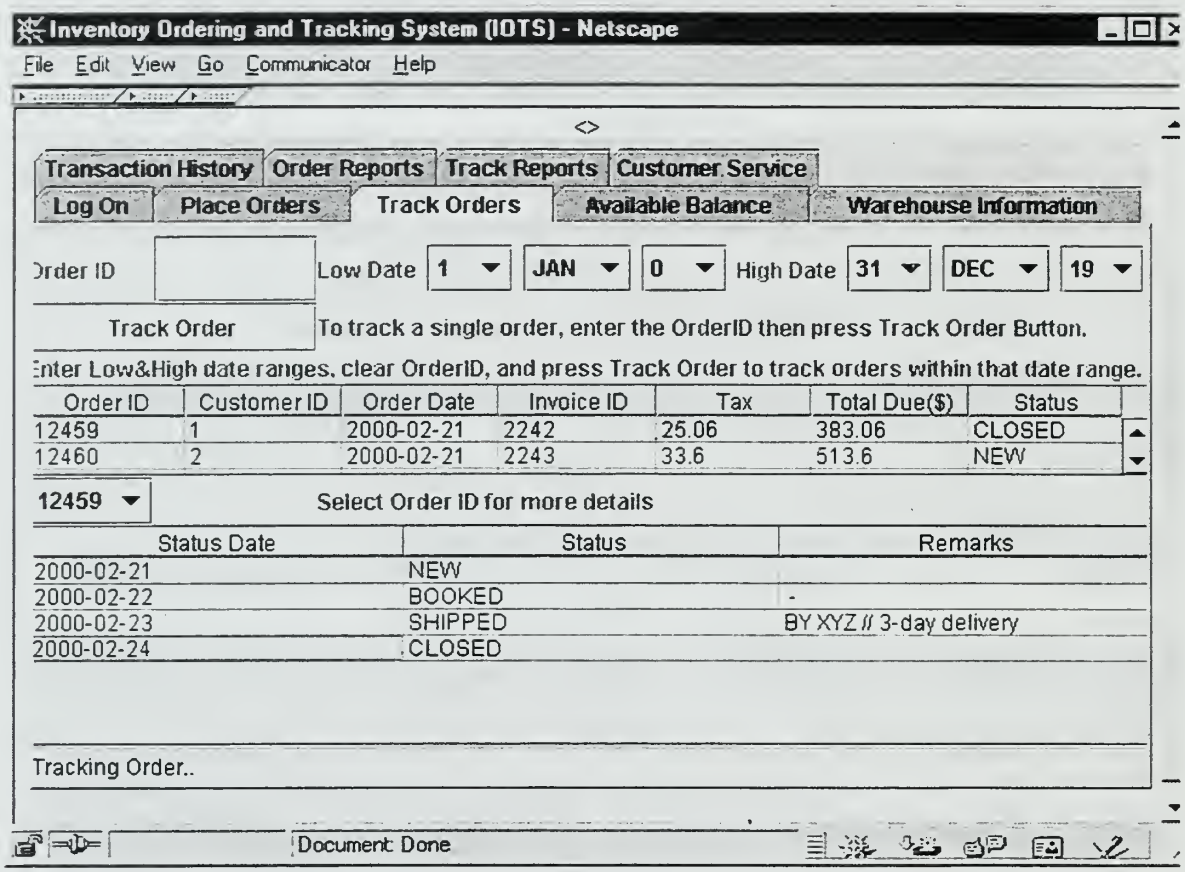


Figure 42. Track Orders Form

E. SUMMARY

This Chapter overviewed the CORBA application design process using Visibroker 3.4 with an illustrative example, defined the QSI's business model, laid out the system architecture and various user interfaces, designed the prototype by utilizing use-case analysis, generated the necessary UML and sequence diagrams, implemented the prototype as a three-tiered client/server application using CORBA/Java, and finally listed and commented on some snapshots of the prototype.

VI. ANALYSIS AND CONCLUSIONS

This chapter addresses the research objectives and research questions that were stated in Chapter I of this thesis. It also lists some of the possible benefits that can be gained from adopting the IOTS over maintaining the status quo. Finally, it presents the conclusions and recommendations.

A. ACHIEVEMENT OF RESEARCH OBJECTIVES AND QUESTIONS

1. Requirements of the New IOTS Database and Application Program for the QSI

Chapter II was completely dedicated to listing the requirements of the new IOTS. In order to determine these requirements, we started by analyzing the current situation. Next, we addressed two major categories of problems that the current IOTS suffers from. The first category consists of the problems imposed by the closed platform and operating system dependent environment that the IOTS operates in. The latter category involves problems related to the way in which the IOTS was designed and implemented.

The major problem that motivates reengineering the current legacy IOTS is its weak supply chain. In the current system, customers are not able to place and track their orders, the QSI is not connected to the suppliers in real-time, and various inventory ordering and tracking activities cannot smoothly interoperate with each other at a system-wide level. Supply chain management integrates both sellers' and buyers' processes to increase the system's responsiveness and competitiveness. The new IOTS will be able to benefit from some of the revolutionary advancements in technology, namely the emergence of Web-based technology, enhancements in the client/server architectures, declines in the price of personal computers, and existence of platform-independent programming languages.

We discussed the business drivers that call for the reengineering process, justify the costs and risks of instantiating this process, and provide the major source of motivation to this research's proposed solution. The requirements were listed to support these business drivers and motivate a solution for both of the problem categories already mentioned. The IOTS prototype was built based on these requirements. The question that was in mind during the design and the implementation of the IOTS prototype is how

implementing these requirements will contribute to the solution of the problems that were identified. Another goal was to figure out what additional benefits can be gained from adopting the modern technology, over just solving the existing problems.

2. Interoperability, Platform and Operating System Dependence

In chapters IV and V of this research, we designed and implemented an object-oriented web-based Inventory Ordering and Tracking System (IOTS) prototype based on the OMG's CORBA. Our goal was to illustrate how CORBA/Java can be used to add value to the business and to address the challenge of interoperability. The challenge of interoperability is one of the most critical problems facing enterprises that want to shift their business to the open market rather than be restricted to a particular operating system or a specific platform. Enterprises need to avoid lock-ins to particular vendor's products and need to develop systems that can work and interact with each other efficiently in the open environment. We achieved our goal by using platform-independent object-oriented programming languages (Java and CORBA) as a backbone that easily connects different applications running on different platforms and different operating systems.

As shown in Figure 43, in the distributed object model offered by CORBA, the client cannot tell whether the target object is local or remote. For example, the IotsApplet, IotsDBClass, and IotsMed client/server objects appear to a client as if they reside on its machine even though these objects exit on different machines and run on top of different operating systems. The user may be aware of where the target object actually resides, but this does not affect the invocation. Therefore the IOTS client and servers implementation may

- operate on different vendors' ORBs;
- exist on different operating systems and platforms;
- work across multiple heterogeneous networks;
- be written in different programming languages,
- and still interoperate as long as clients and servers are implementing the same IDL.

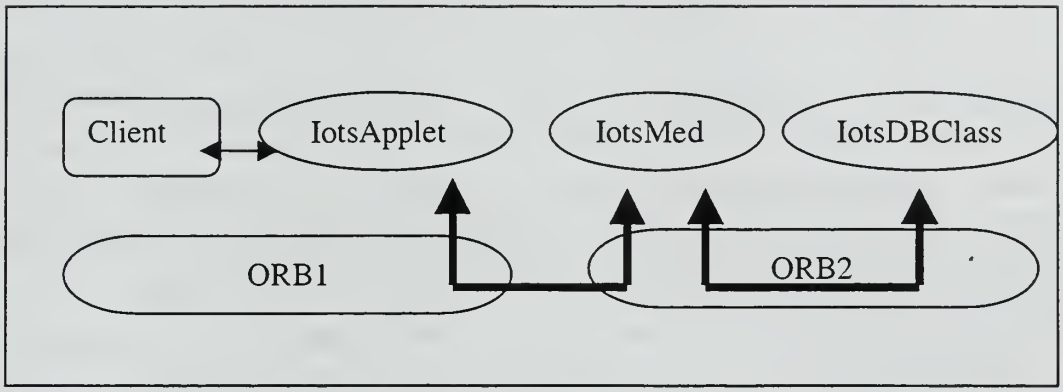


Figure 43. ORB-to-ORB Interoperability

3. Scalability

The three-tier approach adopted by the IOTS prototype allows the QSI to install additional application and database servers as demand increases. As the IOTS grows in size, complexity, and functionality it becomes more necessary to address the issues of load balancing, fault tolerance, and managing components. No distributed system will be able to scale without managing memory, connections, and threads. In the IOTS prototype implementation, we mainly addressed the management of connections. The default threading policy was used, and little more was said about memory management.

Memory management must achieve two major goals. The first is to ensure that the server never leaks memory; that is, once an object is instantiated, it will be eventually freed. The latter is to limit the amount of memory used by a server process. This is accomplished by deactivating idle servant objects. A large number of customers accessing a single server object can be a problem. The following are three different approaches to solving this problem: replicating servers, connection closure, and concentrators. (Slama, 1999, pp. 221-235)

In Chapter V, the Dispenser object was used to assign each customer a single connection object from a pool of live connection objects. This object is released when the customer is no longer using it in order to make it available for other customers. Either the client or the server can perform the connection closure. The IOTS prototype gives the

client the right to decide the connection closure. The connection is closed and the connection object is released and returned to the connection pool when the client closes the application.

The IOTS still needs to address the issues of load balancing and fault tolerance. Load balancing deals with distributing the application load over many servants on many hosts. Fault tolerance is the measure of the degree an application can continue to operate in the presence of failures in hardware/software/OS, physical communication, applications, or a failure due to client-server mismatches. The IOTS prototype provides a limited level of fault tolerance by using the exception classes of Java and CORBA. When an exception occurs, the prototype tries to handle this exception appropriately in order to prevent the whole application from crashing (i.e., continue operation in a degrade mode).

The IOTS is built on top of the Visibroker's ORB. This product also provides some fault tolerance features. In case a connection between a client application and the object implementation fails because of a network error, Visibroker will automatically attempt to re-bind to the server process or a replica of that server. If one of the Visibroker's OSagents becomes unavailable, all object implementations registered with that agent will automatically be registered with another agent.

The issue of scalability in a distributed object environment requires specialized studies; our purpose here is to highlighted the major issues. One of our recommendations is to build a benchmark that tests how scalable the IOTS prototype is when a large number of customers are accessing the CORBA server concurrently.

4. What are the Possible Benefits and Advantages of this Program from Both the Technical and Management Perspectives?

The following are some of the possible benefits if the proposed IOTS is adopted:

- **The QSI is no longer locked into a particular vendor's products**. They can choose their Hardware and Software from among multiple vendors. This will make the procurement process easier and more cost effective. Clearly, having multiple alternatives to choose from enables decision-makers to select the alternative that best suites the enterprise's needs within budget constraints.

- **The QSI can integrate various object components**. With the proposed IOTS, it is easy to integrate purchased object components that model

common knowledge in some business fields and combine them with modules written in-house. For example, the QSI can purchase the executable object implementation along with the IDL file from a vendor. This code is installed on the ORB on the application server. QSI can then write its own client in the language it prefers, independent of language the vendor used to implement the purchased object components.

- **Performance.** Although we used Java to implement both the client and the server in the IOTS prototype, clients and servers do not have to be implemented in the same language. This makes it possible to use each application language in the place where it provides the best performance. For example, Java is very suitable to implement GUIs and Applets so it is one of the best choices to design clients that use the Web. C++ on the other hand can be used to implement the server in order to provide services that require high performance.

  The management of connections, which is used by the IOTS prototype, gives another enhancement to the system's performance. Objects are reserved to clients on demand. Each of these objects run in its own thread and maintains a live connection with the database. Therefore, once one of these objects is allocated to a client, the client does not have to wait for a connection to be established. This will increase the performance and permits the client to be served without much delay. This design, however, increases the overhead and may overload the system if it is being concurrently accessed by a large number of customers. Again, this scalability issue must be handled and the load must be balanced.

- **Object Web**. By adopting the proposed IOTS, the QSI will be deriving benefits from the Object Web. The IOTS prototype client is packaged into a single HTML page that contains several components so that clients can interact with server objects by clicking any of these components without switching the page's context to obtain each response. Thus there is no need to generate dynamically an HTML page for each response. Moreover, the Object Web makes it easy for server objects to callback the client applet. This feature will enable the server to keep the client up-to-date with the most recent information.

- **Reachability.** In the closed environment, the QSI has problems establishing connections with its suppliers, and it supports a limited number of customers. In the open environment, the IOTS enables the QSI to interoperate with the suppliers and customers from all over the globe. Clearly, increasing the number of customers and the market share promises the QSI a more profitable business. There are other non-technical

factors, such as business management, that also play a role in determining the profitability of business.

- **Information Infrastructure.** As discussed in Chapter II, the communication among applications has typically been made through batch processes. This lack of an adequate information infrastructure resulted in redundant data being maintained by different systems and led to a waste of computer resources, manpower, and time. In the proposed IOTS, the communication between all the applications that provide the overall system functionality is performed in real-time, which means that the current system provides more adequate information infrastructure, eliminates potential occurrence of errors, and maintains data integrity.

- **Cost Reduction.** The potential benefits previously mentioned suggest that the QSI might be more profitable by adopting the new IOTS over the closed system it currently operates in, but it is hard to judge by intuition alone whether the new IOTS will provide cost reduction or not. There will be new opportunities for the QSI if it becomes available on the Web. The procurement cost is likely to go down since the QSI will have many choices of technology and will not be locked into any particular vendor's products. Replacing the mainframe with desktop computers also promises additional cost reductions. The cost of maintaining the mainframe is very high and performing the maintenance requires a high-level of expertise usually not available in house. This often must be outsourced, whereas the maintenance cost of desktop computers is far less expensive and can be performed in house. Also it can be performed remotely and automatically, which significantly contributes to cost reduction, at least in terms of manpower. Although putting the power of a large mainframe computer on a desktop is possible, the cost of administering desktop computers that are each configured slightly differently might be significant and outweigh their advantages. The expected savings may not materialize because of unexpected hidden costs; the savings in desktop hardware acquisition are often offset by high annual operating costs for additional labor required to administer and maintain the network. The cost of migrating the current legacy system and the conversion of the IOTS database to work in the new environment may also offset the expected benefits.

5. What is an Appropriate Design for the New IOTS Database and Application Program?

Chapter IV used an iterative methodology to generate an appropriate design for the IOTS database. This methodology consists of the following three steps: 1) define

entities, attributes, and relationships, 2) develop the data model, and 3) transform the data model into a relational database schema. The design was iterative since there is no hard line that separates any two consecutive steps. The further we go in the development process the better understanding of the requirements we get; therefore, there is always a need to go back in order to enhance our design.

In Chapter V, we stepped through the design and the implementation process of the IOTS prototype. We first used Visibroker to go through the application design process with the help of an illustrative example. Next, we designed the IOTS prototype using use-case analysis and the Unified Modeling Language (UML) notation.

6. How Extensible is the New IOTS in Terms of the Use of Different Types of Databases and Different Operating System Environments?

The IOTS was designed with interoperability in mind. The issue of platform and operating system dependence was discussed in the beginning of this chapter. In Chapter IV the IOTS was designed to use a single IOTS database since the requirements did not call for more than one database.

Systems are moving targets. In the future, the QSI may need to connect to more than one database since the business environments keep changing and the initial requirements are likely to change over time. Currently, the CMO centrally manages the inventory levels across all the warehouses. One possible scenario is to decentralize this management process so that the warehouses completely performe it. In this scenario, each warehouse will have its own database, the LSMs will be granted much more authority, and they will be able to choose their suppliers. Meanwhile, the CMO of the QSI will maintain the right to monitor the progress in all of its warehouses and as well as the right to interfere if something goes wrong. Figure 44 shows how a single application may be able to access multiple databases even though these databases belong to different DBMS vendors. The driver manager intermediates the client application program and the DBMS drivers. When the application requests a connection with a database, the driver manager determines which DBMS driver is requested and loads it into memory.

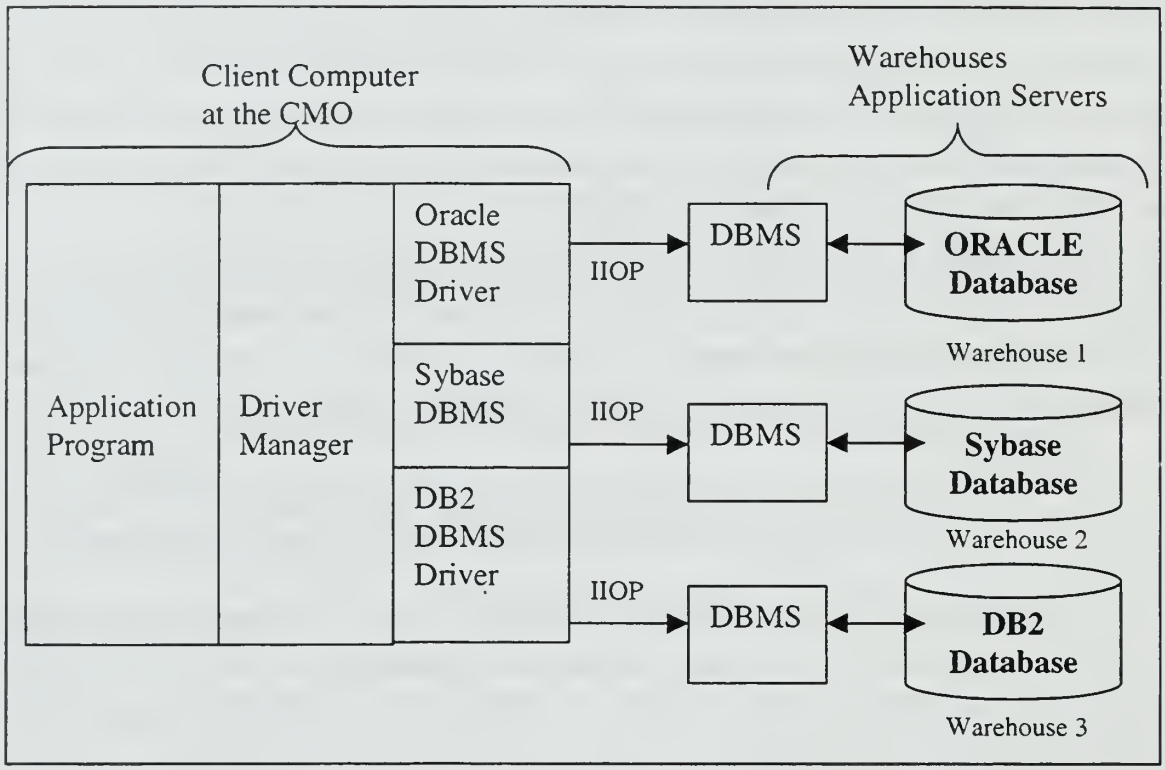


Figure 44. An Application Program Accessing Multiple Databases

The function of the DBMS driver is to receive requests from the application and format and deliver them to the appropriate DBMS. It also receives the responses from the DBMS and formats them for the application. The Open Database Connectivity (ODBC) standard is an interface that can connect to multiple kinds of databases that are ODBC compliant. With ODBC, the QSI can create a single application that connects to databases from different vendors. ODBC drivers must be installed on every client machine. This means that it will be a poor choice for Web-based database systems. In the IOTS prototype, however, we used the Oracle JDBC thin driver rather than the ODBC driver. The JDBC is a Java API that makes it easy to provide a "pure Java" large-scale solutions. The JDBC thin driver can be downloaded on the fly and, therefore, eliminates the need to install it on the client's machine. This approach separates the middle tier from the database on the backend, which makes the system more scalable.

B. CONCLUSIONS

The focus of this research was to design and implement a component-based three-tiered Web-based system prototype that enables our enterprise's applications to interoperate, scale and to exist on different platforms and different operating systems. Component-based distributed systems lower the cost of development and maintenance. Our prototype uses CORBA, an industry-backed, non-proprietary, standard-based distributed architecture and Java, a high-level object-oriented language that enables enterprises to deliver executable code over the Internet. CORBA and Java allow the prototype to run on different operating systems and on multiple platforms. Moreover, the prototype makes it feasible to use different software components from different vendors. These components do not have to be implemented in the same programming language as long as they support the same IDL interface. Therefore, enterprises can mix and match software components according to their needs and will not be locked into a certain vendor.

The prototype has demonstrated how to overcome the stateless nature of the HTTP and build the Object Web by using CORBA and Java. The prototype's client is packaged into a single HTML page that contains several components, thus clients can interact with server objects by clicking any of these components without switching a page's context to obtain each response.

The source code for the prototype can be tailored to specific business requirements. Enterprises, with similar problems to those addressed here, may adopt the development methodology used in this research to solve their own specific problems.

C. AREAS FOR FURTHER RESEARCH

This research can be explored further in the following areas:

- **Scalability and load balancing**. We have used a dispenser object to provide scalability and load balancing. In practice, a Transaction Processing (TP) monitor should be used to handle the interactions between client and server objects. In addition to a TP monitor, we recommend building a benchmark that tests the scalability of the IOTS prototype when large numbers of customers acess the CORBA server at the same time.

- **Security**. The Visibroker SSL package can be purchased and used to provide secure connections between client and server objects.

- **Server Manager(SM).** The Object Web simplifies building an SM tool that monitors and controls all of the objects in the system. One way to build an SM is to register clients, servers, and SM objects with one or more coordinator objects. An SM GUI will use the coordinator to control and monitor the objects registered with it. For example, the SM will be able to monitor what each client is doing and can even terminate clients. The SM can also be designed to set the CORBA callback feature to on or off. If this feature is set, then the server objects will be able to update the clients with the most recent information. For example, An LSMs' display can automatically be updated to reflect the most recent prices without any interference from the user.

- **Benchmarking with RMI, DCOM, and other object-oriented languages**. We have used CORBA as a backbone for our prototype, but different middleware tools, like RMI and DCOM, can be used and benchmarked to show a comparison among different implementations. Server objects can be implemented with C++ or other object-oriented languages instead of Java in order to test whether this implementation provides a significant performance increase over the approach we used in our prototype.

- **Applications vs Applets.** The prototype uses applets as a front end. This approach is suitable for Internet customers where clients download the most recent GUIs. If, however, we have a limited number of LSMs, the software distribution is not a problem, and the LSMs interface does not change frequently, we might consider providing the LSMs with applications not applets. In this approach we sacrifice the automatic upgrade feature of the system for the sake of not downloading the ORB classes every time a client makes a connection with the system.

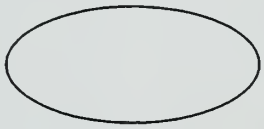
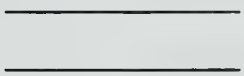
APPENDIX A. ACRONYMS AND TERMS

ACS	Accounting System
API	Application Programming Interface
BOA	Basic Object Adapter
CGI	Common Gateway Interface
CMO	Central Management Office
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DDL	Data Definition Language
DII	Dynamic Invocation Interface
DML	Data Manipulation Language
E-R	Entity Relationship model
ESQL	Embedded Structured Query Language
GIOP	General Inter-ORB Protocol
GSM	Global Store Manager
ICMS	Inventory Control and Management System
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOTS	Inventory Ordering and Tracking System
JDBC	Java Database Connectivity
JDK	Java Development Kit
JVM	Java Virtual Machine
LSM	Local Store Manager
MIS	Management Information System
ODBC	Open Database Connectivity
OMG	Object Management Group
ORB	Object Request Broker
PERS	Personnel System
QSI	Quick Supply Incorporation
SGML	Standard Generalized Markup Language
SM	Schema Manager
SOM	Semantic Object Model
SQL	Structured Query Language
UML	Unified Modeling Language
XML	Extended Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DEMARCO AND YOURDAN'S SYMBOLS FOR DFD'S

This Appendix shows the DeMarco and Yourdon's symbols conventions used in plotting both of the logical and physical DFDs throughout this research.

Symbol	**Meaning**
	Process
	Data store
	Source/ sink
	Data flow

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. LIST OF SOME OF IOTS QUERIES

The following is a list of some of the major queries provided by the IOTS. Although users with different access levels may be allowed to perform the same query, they may not get the same result; the security access levels govern query results, and information is provided on a need-to-know basis:

- List the available balance of a given inventory.
- List the data record of a given inventory.
- List the final status on any given order.
- List a track of all the states on any given order along with the date on which each status took place.
- List the history of any given inventory within a date range.
- List the orders that are Booked, Canceled, Confirmed, Deleted, Waiting, Dues/In, Dues/Out, or shipped for a given inventory in a given store.
- List the orders placed on any given inventory along with their states.
- List the status of all the ordered reports within a date range.
- List the reports that are ready to be picked up.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. LIST OF REPORTS

The following is a list of some of the major reports provided by the system:

1. A report that shows the tracking status for a group of orders. This report shows the consecutive status taken on each order with the date of each status for a given store.

2. A consumption report which shows the items issued to a given store within a date range.

3. A report of Booked, Confirmed, Shipped, Canceled, Deleted, Due-in (D/I), and Due-out (D/O) orders for a given store.

4. A report of the orders waiting for an action.

5. A Statistical report of the number of Booked, Deleted, Confirmed, Shipped, Canceled, Deleted, DI, and DO orders grouped by the originating warehouses. Another similar report must be grouped by the GSMs.

6. A statistical report of the amount of money each GSM spent on purchasing.

7. A report of dead stock inventory which has been bought but not sold for a predefined period of time.

8. A report of the entire inventory available in a given store. This report will show the stock numbers, part numbers, items description, and the available balance of these items.

9. A history report for a group of items that shows all the transactions performed on these items within a date range.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. COMPARING JAVA/CORBA ORBS AND THEIR COMPETITION

The following table shows Orfali and Harkey's rating for the major six Object Web technologies. Four stars represent the highest (best) rating. A blank represents the worst rating; it means the function does not exist.

Feature	CORBA/ IIOP	DCOM	RMI/ RMP	HTTP /CGI	Servlets	Sockets
Abstraction level	****	****	****	**	**	*
Seamless Java Integration	****	****	****	**	**	**
OS platform support	****	**	****	****	****	****
All Java implementation	****	*	****	**	****	****
Typed parameter support	****	****	****	*	*	*
Ease of configuration	***	***	***	***	***	***
Distributed method invocation	****	***	***			
State across invocation	****	***	***		**	**
Dynamic discovery and metadata support	****	***	**		*	
Dynamic invocation	****	****	*			
Performance (remote pings)	*** 3.5 ms	*** 3.8 ms	*** 3.3 ms	827.9	* 55.6 ms	**** 2.1 ms
Wire-level security	****	****	***	***	***	***
Wire-level transactions	****	***				
Persistence object references	****	*				
URL-based naming	****	**	**	****	****	***
Multilingual object invocations	****	****		***		****
Language-neutral wire protocol	****	****		****	****	
Intergalactic scaling	****	**	*	**	**	****
Open standard	****	**	**	****	**	****

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. ENTERPRISE LEVEL DATA SUBJECT DEFINITION

Data Subject Name	Data Subject Definition
ITEM	Contains general information about all items, such as item name, component identification, part number, unit of sale, item price, etc.
CUSTOMER	Anyone who contacts the QSI to place an order. Contact might be performed by using web-browsers, email, telephone, fax, regular mail, in person, etc.
INVENTORY	Contains the inventory levels of all items in all warehouses.
INVOICE	Contains information about each invoice billed to a customer
ORDERS	An order is any request for an item or items offered for sale in the business of the QSI. The LSMs can also order reports from the CMO.
TRANSACTION	Each single transaction has a unique code, description, and effect on inventory levels.
USERS	This data subject contains the authorized users who have access to the IOTS. It contains LSMs, GSMs, customers, and MIS users. This data subject contains the users' Security Access Levels. Although the DBMS will take care of security, this data subject is required to provide security on the application level.
WAREHOUSE	Keeps a list of all the necessary information about all QSI's warehouses. For each warehouse, this data subject will include the name, address, phone(s), fax(s), email, etc...

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. ENTERPRISE LEVEL ATTRIBUTE DEFINITION

This Appendix contains a list of candidate attributes that will be used in the IOTS database:

Attribute	Definition
Balance	Available quantity
ItemID	Component identification number
CreditCardExpDate	Credit card expiration date
CreditCardNumber	Credit card number
CreditCardType	Credit card type
CustomerFirstName	Customer's first name
CustomerID	Customer's identification number
CustomerLastName	Customer's last name
EmailAddr	Email address
FaxNumber	Fax number
InvoiceDate	Invoice date
InvoiceID	Invoice identification
InvoiceTotal	Invoice total amount due
ItemDescription	Component name
OrderDateAndTime	Date and time of the order placement
OrderID	Order identification number
PartNumber	Component part number
PurchasePrice	Purchase price of an item
Quantity	Quantity ordered
Remarks	A note that explains why an order is deleted of canceled
ReorderLevel	When the reorder level quantity is reached, the warehouse places a new order to make this inventory available for future requests.
ReorderQty	The quantity ordered when the ReorderLevel quantity is reached
ResAddrCity	City of residence address
ResAddrState	State of residence address
ResAddrStreet	Street of residence address
ResAddrZip	Zip code of residence address
SalePrice	Price of sale
ShipAddrCity	City of shipping address
ShipAddrState	State of shipping address
ShipAddrStreet	Street of shipping address
ShipddrZip	Zip code of residence address
Attribute	**Definition**

Status	Order's status
StatusDateAndTime	Date and Time on which a status took place
Tax	Tax amount due
TaxableItem	Boolean flag contains true if the item is taxable; otherwise it contains false
TelephoneNumber	Telephone number
TotalDue	Total amount due for an order
TransactionCode	The code of a transaction
TransactionDesc	The description of a transaction
UnitOfPurchase	Unit of purchase (ex EA, PK, DZ, etc)
UnitOfSale	Unit of sale
UserAccessLevel	User access level (L0, L1, L2, L3, and L4)
UserID	User log on identification
UserName	The complete name of the user
UserPassword	The password of the user (Stored encrypted)
WareAddrCity	City of warehouse address
WareAddrState	State of warehouse address
WareAddrStreet	Street of warehouse address
WareAddrZip	Zip code of warehouse address
WarehouseID	Warehouse identification code
WarehouseName	Name of the Warehouse

APPENDIX H. ENTERPRISE LEVEL DATA SUBJECTS' ATTRIBUTES

This Appendix lists the attributes of each data subject:

Data Subject Name	Entity Type	Attributes
ITEM	Independent	ItemID, ItemDescription, PartNumber, PurchasePrice, UnitOfPurchase, SalePrice, UnitOfSale, TaxableItem
CUSTOMER	Independent	CustomerID, CustomerName, ResAddrStreet, ResedenceAddress, Shipping Address, TelephoneNumber, FaxNumber, EmailAddress, CreditCardNumber, CreditCardExpDate, CreditCardType
INVENTORY	Dependent	InventoryID, Balance, ReorderLevel, ReorderQty
INVOICE	Dependent	InvoiceID, InvoiceDate, ORDERS, InvoiceTotal
ORDERS	Dependent	OrderID, OrderDateAndTime, CustomerID, LineItem (ItemID, Quantity, UnitPrice, ExtendedPrice), Tax, TotalDue, StatusTrackInfo (StatusDateAndTime, Status, Remarks)
TRANSACTION	Independent	TransactionCode, TransactionDesc
USERS	Independent	UserID, UserName, UserAccessLevel
WAREHOUSE	Independent	WarehouseID, InventoryID, WarehouseName, WarehouseAddress, FaxNumber, TelephoneNumber

THIS PAGE INTENTIONALLY LEFT BLANK

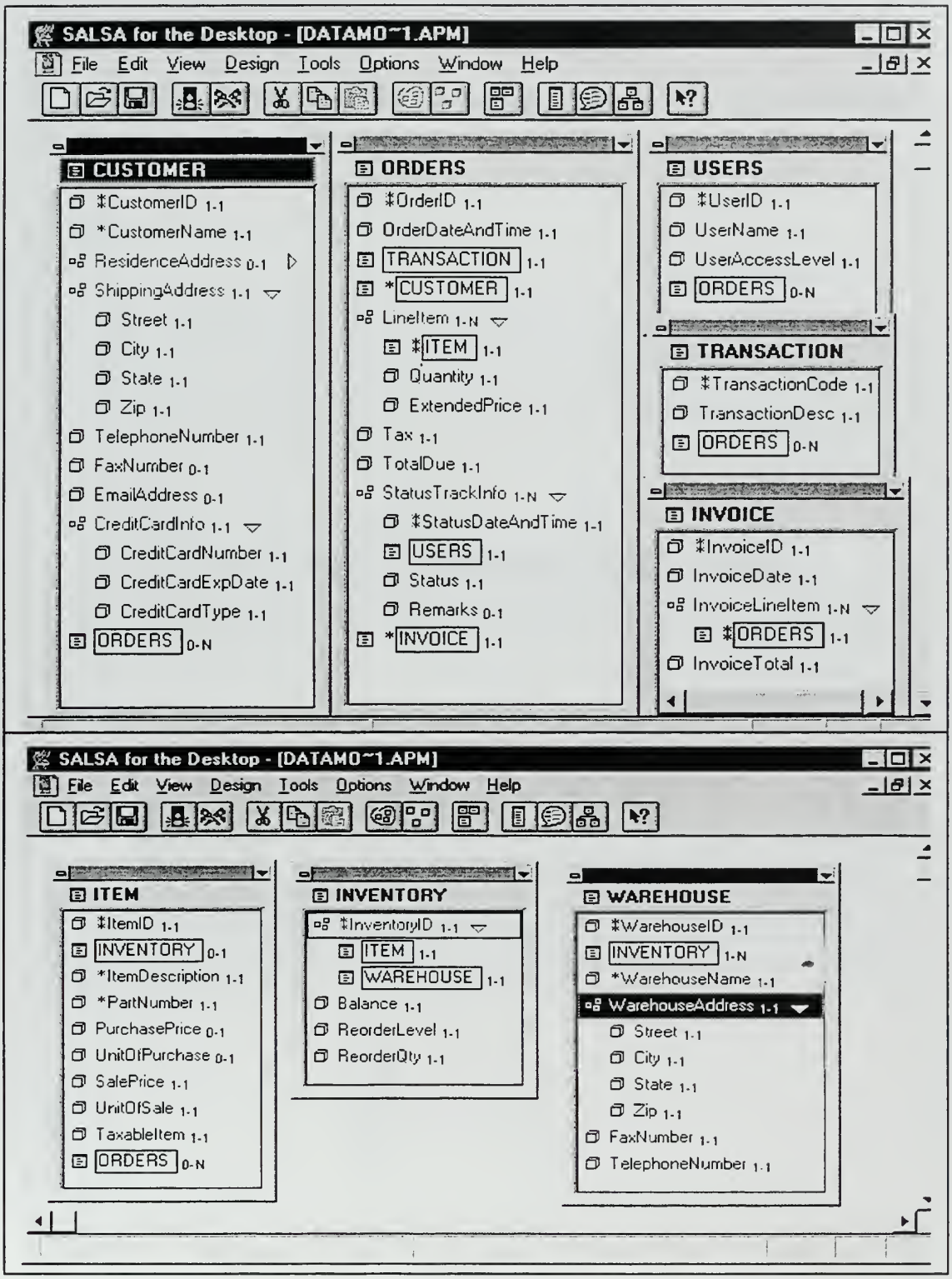
APPENDIX I. ENTERPRISE LEVEL RELATIONSHIP DEFINITION

Parent data subject	Child data subject	Relationship Definition
ITEM	INVENTORY	- An item may or may not have an inventory record (an inventory record is the inventory levels of a certain item in a certain store) - If an inventory record exists, it must reference an existing item record.
INVOICE	ORDERS	- An invoice must contain at least one order. - An order must appear on one and only one invoice.
ITEM	ORDERS	- An item may never be ordered, or it can be ordered many times. - An order may contain multiple different items, but a single order may not contain the same item more than once. - For an order to be valid, it must contain at least one item.
CUSTOMER	ORDERS	- A customer may place one to many orders, or may not place orders at all. - A single order must be associated with one and only one customer.
TRANSACTION	ORDERS	- Each order must have one and only one transaction type (examples of transaction types: ordered by customer, ordered by a warehouse, etc) - A transaction type may appear on more than one order
USERS	ORDERS	- A user may change the status of a single order more than once, or may not make any change - A state change must be associated with one and only one user code.
WAREHOUSE	INVENTORY	- An item may exist in multiple warehouses with different inventory level in each different warehouse. - A warehouse tracks the inventory levels of many items.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX J. SEMANTIC DATA MODEL FOR THE IOTS

This appendix lists the IOTS semantic data model developed using "SALSA for disk top" modeling tool



THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K. ENTERPRISE LEVEL ENTITY DEFINITION

Relation	Attribute	Column Datatype	Null Option	Primary Key	Foreign Key
ITEM	ItemID	VARCHAR2(13)	NOT NULL	YES	
	ItemDescreption	VARCHAR2(30)	NOT NULL		
	PartNumber	VARCHAR2(25)	NOT NULL		
	PurchasePrice	NUMBER(8,2)	NULL is OK		
	UnitOfPurchase	CHAR(2)	NULL is OK		
	SalePrice	NUMBER(8,2)	NOT NULL		
	UnitOfSale	CHAR(2)	NOT NULL		
	TaxableItem	BOOLEAN	NOT NULL		
CUSTOMER	CustomerID	VARCHAR2(13)	NOT NULL		
	CustomerName	VARCHAR2(30)	NOT NULL	YES	
	ResAddrStreet	VARCHAR2(50)	NOT NULL		
	ResAddrCity	VARCHAR2(25)	NULL is OK		
	ResAddrState	CHAR(2)	NULL is OK		
	ResAddrZip	CHAR(10)	NULL is OK		
	ShipAddrStreet	VARCHAR2(50)	NOT NULL		
	ShipAddrCity	VARCHAR2(25)	NOT NULL		
	ShipAddrState	CHAR(2)	NOT NULL		
	ShipAddrZip	CHAR(10)	NOT NULL		
	TelephoneNumber	CHAR(12)	NOT NULL		
	FaxNumber	VARCHAR2(12)	NULL is OK		
	EmailAddress	VARCHAR2(35)	NULL is OK		
	CreditCardNumber	CHAR(16)	NOT NULL		
	CreditCardExpDate	DATE	NOT NULL		
	CreditCardType	VARCHAR2(20)	NOT NULL		

Relation	Attribute	Column Datatype	Null Option	Primary Key	Foreign Key
INVENTORY	ItemID-FK	VARCHAR2(13)	NOT NULL	YES	YES
	WarehouseID-FK	VARCHAR2(5)	NOT NULL	YES	YES
	Balance	NUMBER(10,2)	NOT NULL		
	ReorderLevel	NUMBER(10,2)	NOT NULL		
	ReorderQty	NUMBER(10,2)	NOT NULL		
ORDERS	OrderID	VARCHAR2(13)	NOT NULL	YES	
	OrderDateAndTime	DATE	NOT NULL		
	CustomerID-FK	VARCHAR2(13)	NOT NULL		YES
	TransactionCode-FK	CHAR(3)	NOT NULL		YES
	Tax	NUMBER(10,2)	NOT NULL		
	TotalDue	NUMBER(10,2)	NOT NULL		
	InvoiceID-FK	VARCHAR(13)	NOT NULL		YES
ORDERS_DETAILS	OrderID-FK	VARCHAR2(13)	NOT NULL	YES	YES
	ItemID-FK	VARCHAR2(13)	NOT NULL	YES	YES
	Quantity	NUMBER(10,2)	NOT NULL		
	UnitPrice	NUMBER(8,2)	NOT NULL		
	ExtendenPrice	NUMBER(10,2)	NOT NULL		
ORDERS_STATUS	OrderID-FK	VARCHAR2(13)	NOT NULL	YES	YES
	StatusDateAndTime	DATE	NOT NULL	YES	
	UserID-FK	VARCHAR2(13)	NOT NULL		YES
	Status	VARCHAR2(20)	NOT NULL		
	Remarks	VARCHAR2(50)	NULL is OK		
TRANSACTION	TransactionCode	CHAR(3)	NOT NULL	YES	
	TransactionDesc	VARCHAR2(50)	NOT NULL		
USERS	UserID	VARCHAR2(13)	NOT NULL	YES	
	UserName	VARCHAR2(30)	NOT NULL		
	UserAccessLevel	CHAR(2)	NOT NULL		

Relation	Attribute	Column Datatype	Null Option	Primary Key	Foreign Key
WAREHOUSE	WarehouseID	VARCHAR2(5)	NOT NULL	YES	
	WarehouseName	VARCHAR2(35)	NOT NULL		
	WareAddrStreet	VARCHAR2(50)	NOT NULL		
	WareAddrCity	VARCHAR2(25)	NOT NULL		
	WareAddrState	CHAR(2)	NOT NULL		
	WareAddrZip	CHAR(10)	NOT NULL		
	FaxNumber	CHAR(12)	NOT NULL		
	TelephoneNumber	CHAR(12)	NOT NULL		
INVOICE	InvoiceID	VARCHAR2(13)	NOT NULL	YES	
	InvoiceDate	DATE	NOT NULL		
	InvoiceTotal	NUMBER(10,2)	NOT NULL		
INVOICE_DETAILS	InvoiceID-FK	VARCHAR2(13)	NOT NULL	YES	YES
	OrderID-FK	VARCHAR2(13)	NOT NULL	YES	YES

137

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX L. DATABASE DEFINITION

This Appendix lists the IOTS database definition written in the SQL DDL. It defines the tables, indexes, attributes, primary keys, foreign keys, and constraints.

```
/* ******************************************************** */
/*
REM // struct.txt contains IOTS database structure definition
REM // last modified 02/17/2000
*/
/* ******************************************************** */

REM Table: ITEM

DROP   TABLE ITEM CASCADE CONSTRAINTS;
CREATE TABLE ITEM (
        ItemID              VARCHAR2(13)     PRIMARY KEY,
        ItemDescreption     VARCHAR2(30)     NOT NULL,
        PartNumber          VARCHAR2(25)     NOT NULL,
        PurchasePrice       NUMBER(8,2),
        UnitOfPurchase      CHAR(2),
        SalePrice           NUMBER(8,2)      NOT NULL,
        UnitOfSale          CHAR(2)          NOT NULL,
        TaxableItem         CHAR(1)          NOT NULL
);

/* ******************************************************** */

REM Table: CUSTOMER

DROP   TABLE CUSTOMER CASCADE CONSTRAINTS;
CREATE TABLE CUSTOMER (
        CustomerID          VARCHAR2(13)     PRIMARY KEY,
        CustomerName        VARCHAR2(30)     NOT NULL,
        ResAddrStreet       VARCHAR2(50)     NOT NULL,
        ResAddrCity         VARCHAR2(25),
        ResAddrState        CHAR(2),
        ResAddrZip          CHAR(10),
        ShipAddrStreet      VARCHAR2(50)     NOT NULL,
        ShipAddrCity        VARCHAR2(25)     NOT NULL,
        ShipAddrState       CHAR(2)          NOT NULL,
        ShipAddrZip         CHAR(10)         NOT NULL,
        TelephoneNumber     CHAR(12)         NOT NULL,
        FaxNumber           VARCHAR2(12),
```

```
        EmailAddress        VARCHAR2(35),
        CreditCardNumber    CHAR(16)          NOT NULL,
        CreditCardExpDate   DATE              NOT NULL,
        CreditCardType      VARCHAR2(20)      NOT NULL );

/* ********************************************************** */

REM Table: WAREHOUSE

DROP  TABLE WAREHOUSE CASCADE CONSTRAINTS;
CREATE TABLE WAREHOUSE (
        WarehouseID         VARCHAR2(5)       PRIMARY KEY,
        WarehouseName       VARCHAR2(35)      NOT NULL,
        WareAddrStreet      VARCHAR2(50)      NOT NULL,
        WareAddrCity        VARCHAR2(25)      NOT NULL,
        WareAddrState       CHAR(2),
        WareAddrZip         CHAR(10)          NOT NULL,
        FaxNumber           CHAR(12)          NOT NULL,
        TelephoneNumber     CHAR(12)          NOT NULL
);

/* ********************************************************** */

REM Table: INVENTORY

DROP  TABLE INVENTORY CASCADE CONSTRAINTS;
CREATE TABLE INVENTORY (
        ItemID_FK           VARCHAR2(13)      REFERENCES ITEM,
        WarehouseID_FK      VARCHAR2(5)       REFERENCES WAREHOUSE,
        Balance             NUMBER(10,2)      NOT NULL,
        ReorderLevel        NUMBER(10,2)      NOT NULL,
        ReorderQty          NUMBER(10,2)      NOT NULL,
        PRIMARY KEY (ItemID_FK, WarehouseID_FK)
);

/* ********************************************************** */

REM TABLE: INVOICE

DROP  TABLE INVOICE CASCADE CONSTRAINTS;
CREATE TABLE INVOICE (
        InvoiceID     VARCHAR2(13)      PRIMARY KEY,
        InvoiceDate   DATE              NOT NULL,
```

```
        InvoiceTotal    NUMBER(10,2)        NOT NULL
);

REM TABLE: INVOICE_DETAILS

DROP   TABLE INVOICE_DETAILS CASCADE CONSTRAINTS;
CREATE TABLE INVOICE_DETAILS (
        InvoiceID_FK            VARCHAR2(13)      REFERENCES INVOICE,
        OrderID_FK              VARCHAR2(13)      REFERENCES ORDERS,
      PRIMARY KEY (InvoiceID_FK, OrderID_FK)
);

/* ********************************************************** */

REM  TABLE: TRANSACTION

DROP   TABLE TRANSACTION CASCADE CONSTRAINTS;
CREATE TABLE TRANSACTION (
        TransactionCode         CHAR(3)             PRIMARY KEY,
        TransactionDesc         VARCHAR2(50)        NOT NULL
);

/* ********************************************************** */

REM  TABLE: USERS

DROP   TABLE USERS CASCADE CONSTRAINTS;
CREATE TABLE USERS (
        UserID                  VARCHAR2(13)      PRIMARY KEY,
        UserName                VARCHAR2(30)      NOT NULL,
        UserAccessLevel         CHAR(2)           NOT NULL
);

/* ********************************************************** */

REM  TABLE: ORDERS

DROP   TABLE ORDERS CASCADE CONSTRAINTS;
CREATE TABLE ORDERS (
        OrderID                 VARCHAR2(13)      PRIMARY KEY,
        CustomerID_FK           VARCHAR2(13)      REFERENCES CUSTOMER,
        OrderDateAndTime        DATE              NOT NULL,
        TransactionCode_FK CHAR(3)                REFERENCES TRANSACTION,
```

```
        Tax             NUMBER(10,2)      NOT NULL,
        TotalDue        NUMBER(10,2)      NOT NULL,
    InvoiceID_FK        VARCHAR2(13)      REFERENCES INVOICE
);

REM TABLE: ORDER_DETAILS

DROP TABLE ORDER_DETAILS CASCADE CONSTRAINTS;
CREATE TABLE ORDER_DETAILS (
        OrderID_FK  VARCHAR2(13)      REFERENCES ORDERS,
        ItemID_FK   VARCHAR2(13)      REFERENCES ITEM,
        Quantity    NUMBER(10,2)      NOT NULL,
        UnitPrice   NUMBER(08,2)      NOT NULL,
        ExtendedPrice NUMBER(10,2)    NOT NULL,
    PRIMARY KEY(OrderID_FK, ItemID_FK)
);

REM TABLE: ORDER_STATUS

DROP TABLE ORDER_STATUS CASCADE CONSTRAINTS;
CREATE TABLE ORDER_STATUS (
        OrderID_FK          VARCHAR2(13)      REFERENCES ORDERS,
        StatusDateAndTime   DATE              NOT NULL,
        UserID_FK           VARCHAR2(13)      REFERENCES USERS,
        Status              VARCHAR2(20)      NOT NULL,
        Remarks             VARCHAR2(50),
        PRIMARY KEY(OrderID_FK, StatusDateAndTime, UserID_FK)
);
/* ********************************************************** */
```

APPENDIX M. IDL SPECIFICATIONS FOR THE IOTS PROTOTYPE

This Appendix contains the IDL file used to define the interface among the IOTS CORBA objects:

```
module Iots
{
 struct customerStruct
  {
   string customerID;
   string name;
   string bStreet;
   string bCity;
   string bState;
   string bZipCode;
   string sStreet;
   string sCity;
   string sState;
   string sZipCode;
   string telephone;
   string phax;
   string email;
   string creditCardNo;
   string expDate;
   string creditCardType;
   };
   typedef customerStruct cusStruct;

  struct warehouseStruct
  {
   string sWarehouseID;
   string sWarehouseName;
   string sWareAddrStreet;
   string sWareAddrCity;
   string sWareAddrState;
   string sWareAddrZip;
   string sFaxNumber;
   string sTelephoneNumber;
  };
  typedef warehouseStruct wareStruct;

  struct balanceStruct
  {
   string sItemID;
   string sItemDescreption;
   string sPartNumber;
   string sTaxableItem;
```

```
  float  sSalePrice;
  string sUnitOfSale;

  string sWarehouseID;
  string sWarehouseName;

  float  sBalance;
  float  sReorderLevel;
  float  sReorderQty;
 };
 typedef balanceStruct    balStruct;

struct IDsStruct {
  string sWareID;
};
typedef sequence<IDsStruct> wareIDs;

struct ItemIDsStruct {
  string sItemID;
};
typedef sequence<ItemIDsStruct> itemIDs;

struct ItemIDsStruct2 {
  string sItemID;
  string sItemDescreption;
  float  sSalePrice;
};
typedef sequence<ItemIDsStruct2> itemIDs2;

struct TransactionStruct {
  string sTransactionCode;
  string sTransactionDesc;
};
typedef sequence<TransactionStruct> transIDs;

struct CustomerIDsStruct {
  string sCustomerID;
  string sCustomerName;
};
typedef sequence<CustomerIDsStruct> cusIDs;

struct OrderStruct {
    string itemID;
    string itemDescreption;
    long   quantity;
    float  salePrice;
    float  extendedPrice;
```

```
    float  tax;
    float  total;
};
typedef sequence<OrderStruct> ordStruct;

struct OrderResultStruct{
    string sOrderID;
    string sMessage;
};
typedef OrderResultStruct ordResult;

struct StatStruct {
    string sUserID;
    string sTransactionCode;
    string sCustomerID;
    string sOrderDateAndTime;
    float  sTaxAll;
    float  sTotalDue;
};
typedef StatStruct stStruct;

struct OrderMainInfoStruct {
   string sOrderID;
   string sCustomerID;
   string sTax;
   string sTotalDue;
   string sOrderDateAndTime;
   string sInvoiceID;
   string sStatus;
};
typedef sequence<OrderMainInfoStruct> mainInfo;

struct StatusStruct {
   string sStatusDate;
   string sStatus;
   string sRemarks;
};
typedef sequence<StatusStruct> statseq;

interface IotsMed
 {
   void addCustomer(in customerStruct cusStruct);
   cusStruct getCustomer(in string cusID);
   void modCustomer(in customerStruct cusStruct);

   void addWarehouse(in warehouseStruct wareStruct);
   wareStruct getWarehouse(in string wareID);
```

```
    void modWarehouse(in warehouseStruct wareStruct);

    balStruct getBalance(in string cusID, in string
    wareID);

    wareIDs   getWareIDs();
    itemIDs   getItemIDs();
    itemIDs2  getItemIDs2();

    boolean   connect(in string user, in string password);

    transIDs  getTransIDs();
    cusIDs    getCusIDs();

    string    getDate();

    ordResult placeOrder(in ordStruct inOrder, in stStruct
    stat);

    mainInfo getOrderMainInfo(
                  in string orderID, in string lDate, in
     string   hDate );
    statseq  getStatus (in string orderID);
  };

  interface IotsDispenser
  {
    IotsMed reserveObject();
    void    releaseObject(in IotsMed iotsmed);
  };
};
```

APPENDIX N. INDEX.HTML CODE

This Appendix contains the HTML code for the Index.htm. This code was built using Microsoft Frontpage 98 and was used to embed the applet "IotsApplet .class".

```
<html>
<head>
<meta name="GENERATOR" content="Microsoft FrontPage 3.0">
<meta name="Microsoft Theme" content="none, default">
<title>Inventory Ordering and Tracking System (IOTS)</title>
<meta name="Microsoft Border" content="none, default">
</head>
<body bgcolor="#FFFFFF">
<h1 align="center"><img src="flower.jpg" width="38" height="55"
alt="flower.wmf (8182 bytes)" align="left"><font color="#FF0000">
            
   
Quick Supply Inc.</font>
            
   
<img src="AG00001_.gif" width="39" height="47" alt="AG00001_.gif (4995 bytes)">
</h1>
<h1 align="center"><font color="#FF0000" size="4">LSMs Cotrol Page </font><img
src="WB01385_.gif" width="500" height="4" alt="WB01385_.gif (6261 bytes)"></h1>
<p align="center">
<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" WIDTH="301"
HEIGHT="90"
codebase="http://java.sun.com/products/plugin/1.2/jinstall-12-
win32.cab#Version=1,2,0,0"
```

```
align="left">
  <param name="CODE" value="IotsApplet.class">
  <param name="type" value="application/x-java-applet;version=1.2">
  <param name="org.omg.CORBA.ORBClass" value="com.visigenic.vbroker.orb.ORB">
  <param name="ORBgatekeeperIOR" value="http://computerb:15000/gatekeeper.ior">
  <param name="ORBalwaysProxy" value="true">
  <param name="ORBservices" value="CosNaming">
  <param name="SVCnameroot" value="IotsNames"><> <embed type="application/x-
java-applet;version=1.2" java_CODE="IotsApplet.class"
WIDTH="600" HEIGHT="350"
org.omg.CORBA.ORBClass="com.visigenic.vbroker.orb.ORB"
ORBgatekeeperIOR="http://computerb:15000/gatekeeper.ior" ORBalwaysProxy="true"
ORBservices="CosNaming" SVCnameroot="IotsNames"
pluginspage="http://java.sun.com/products/plugin/1.2/plugin-
install.html"><noembed></COMMENT>
</noembed>
</object>
</p>
<!--
<APPLET  CODE = IotsApplet.class WIDTH = 600 HEIGHT = 350 >
<PARAM NAME = org.omg.CORBA.ORBClass VALUE
=com.visigenic.vbroker.orb.ORB>
<PARAM NAME = ORBgatekeeperIOR VALUE
=http://computerb:15000/gatekeeper.ior>
<PARAM NAME = ORBalwaysProxy VALUE =true>
<PARAM NAME = ORBservices VALUE =CosNaming>
<PARAM NAME = SVCnameroot VALUE =IotsNames>
</APPLET> -->
<!--"END_CONVERTED_APPLET"-->
</body>
</html>
```

APPENDIX O. SOFTWARE COMPONENTS USED IN THE IOTS

The following table lists the various software components and development tools used to design and implement the IOTS prototype:

Category	Tool Name	Purpose
Development and Deployment Tools	Jvision	To draw UML diagrams for the IOTS prototype.
	SQL* Plus	To pass SQL commands to the IOTS database on the back end for testing purposes
	SALSA	To develop the IOTS Semantic Object Model
	JDK1.2.2	To develop the IOTS Java classes
	Microsoft Frontpage 98	To develop the main page of the IOTS prototype that hosts the IOTS's applet.
	Visigenic Visibroker 3.4	To develop and deploy the IOTS Java classes
	JVM Plug-ins 1.2.2	To enable Web browsers to override their old backed-in JVMs. These plug-ins will make it possible for Web browsers to execute the prototype that was developed with JDK 1.2.2
	ORACLE8i	To build the IOTS database on the back end
	Microsoft Internet Information Server (IIS)	To provide HTTP service
	Inprise Gatekeeper 3.3	To act as a proxy server that overcomes some of the applet's limitations
	ORACLE JDBC thin driver	To connect the middle tier with the IOTS database on the back end
	Netscape and Microsoft Internet explorer	Used for testing purposes

Category	Tool Name	Purpose
IOTS Prototype	IotsApplet, IotsLSMsBean, IotsServer, IotsDispenserImpl (by Orfali and Harkey), IotsMedImpl, IotsDBClass, SecurityBean, WarehouseInfoBean, CustomerBean, PlaceOrderBean, TrackOrderBean, BalanceBean, Iots.Idl, index.htm	To provide the IOTS prototype with the required functionalities

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX P. SOURCE CODE OF THE IOTS PROTOTYPE

This Appendix lists the source code of the IOTS prototype. It lists the code for the following classes:

- **IotsApplet**
- **IotsServer**
- **IotsLSMsBean**
- **IotsDispenserImpl**
- **IotsMedImpl**
- **IotsDBClass**
- **SecurityBean**
- **WarehouseInfoBean**
- **CustomerBean**
- **PlaceOrderBean**
- **TrackOrderBean**
- **BalanceBean**

A. IOTSAPPLET

```
//IotsApplet.java
import java.awt.*;
import javax.swing.*;
import java.applet.*;
import IotsLSMsBean;
public class IotsApplet extends JApplet
{
 private IotsLSMsBean IotsLSMsBean1;
  public void init()
  {
    getContentPane().setLayout(new BorderLayout(0,0));
    setSize(600,350);
    IotsLSMsBean1 = new IotsLSMsBean(this);
    IotsLSMsBean1.setBackground(java.awt.Color.orange);
    getContentPane().add("Center",IotsLSMsBean1);
  }//init
```

```
 public void destroy()
 {
  IotsLSMsBean1.releaseDBObject();
 }
}//IotsApplet
```

B. IOTSSERVER

```
// IotsServer.java
import org.omg.CosNaming.*;
public class IotsServer
{ static public void main(String[] args)
 {
  int numberInstances;
  try
  {
   if (args.length == 0)
     numberInstances = 4;
   else
     numberInstances = Integer.parseInt(args[0]);
   // Initialize the ORB
   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
   // Create the IotsDispenser object
   IotsDispenserImpl dispenser =
   new IotsDispenserImpl(args, "My Dispenser", numberInstances);
   // Export the newly create object
   orb.connect(dispenser);
   // Get a reference to the Naming service
   org.omg.CORBA.Object nameServiceObj =
          orb.resolve_initial_references ("NameService");
   if (nameServiceObj == null)
   {
    System.out.println("nameServiceObj = null");
    return;
   }
   org.omg.CosNaming.NamingContext nameService =
      org.omg.CosNaming.NamingContextHelper.narrow(nameServiceObj);
   if (nameService == null)
   {
    System.out.println("nameService = null");
    return;
```

```
    }
    // bind the IotsMed object in the Naming service
    NameComponent[] IotsName = {new NameComponent("IotsMed", "")};
    nameService.rebind(IotsName, dispenser);
    // wait forever for current thread to die
    Thread.currentThread().join();
   } catch(Exception e)
   { System.err.println(e);
   }
  }
}//IotsServer
```

C. IOTSLSMSBEAN

```
//IotsLSMsBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.Applet;
import org.omg.CosNaming.*;
public class IotsLSMsBean extends JPanel{
 private JTabbedPane jtp;
 private org.omg.CosNaming.NamingContext nameService;
 private Iots.IotsDispenser        myDispenser;
 private Iots.IotsMed              myIotsMed;
 private CustomerBean              customerBean;
 private WarehouseInfoBean         warehouseInfoBean;
 private BalanceBean               balanceBean;
 private SecurityBean              securityBean;
 private PlaceOrderBean            placeOrderBean;
 private TrackOrderBean            trackOrderBean;
 public IotsLSMsBean(JApplet japp) {
  setSize(600,350);
  initializeORB(japp);
  setLayout(new BorderLayout());
  jtp = new JTabbedPane();
  JPanel placeOrdersPanel       = new JPanel();
   placeOrdersPanel.add(
    placeOrderBean= new PlaceOrderBean(myIotsMed));
  JPanel trackOrdersPanel       = new JPanel();
   trackOrdersPanel.add(
```

```
    trackOrderBean   = new TrackOrderBean(myIotsMed));
  JPanel availableBalancePanel   = new JPanel();
   availableBalancePanel.add(
    balanceBean= new BalanceBean(myIotsMed));
  JPanel warehouseInfoPanel      = new JPanel();
   warehouseInfoPanel.add(
    warehouseInfoBean = new WarehouseInfoBean(myIotsMed));
  //History, order and track reports are not implemented
  JPanel historyPanel            = new JPanel();
  JPanel orderReportPanel        = new JPanel();
  JPanel trackReportsPanel       = new JPanel();
  JPanel customerPanel           = new JPanel();
   customerPanel.add(
    customerBean = new CustomerBean(myIotsMed));
  JPanel securityPanel           = new JPanel();
   securityPanel.add(
    securityBean = new SecurityBean(myIotsMed,myDispenser));
  jtp.addTab("Log On", securityPanel);
  jtp.addTab("Place Orders", placeOrdersPanel);
  jtp.addTab("Track Orders", trackOrdersPanel);
  jtp.addTab("Available Balance", availableBalancePanel);
  jtp.addTab("Warehouse Information", warehouseInfoPanel);
  jtp.addTab("Transaction History", historyPanel);
  jtp.addTab("Order Reports", orderReportPanel);
  jtp.addTab("Track Reports", trackReportsPanel);
  jtp.addTab("Customer Service", customerPanel);
  // set background colors
  for (int i=0; i < 9; i++)
  {
   jtp.setBackgroundAt(i,java.awt.Color.green);
  }
  add("Center",jtp);
  setVisible(true);
}
public void initializeORB(JApplet app)
 {
  try
  { // Initialize the ORB
    org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(app, null);
    // Get a reference to the Naming service
    org.omg.CORBA.Object nameServiceObj =
      orb.resolve_initial_references ("NameService");
    if (nameServiceObj == null)
```

```
    {
     System.out.println("nameServiceObj = null");
     return;
    }
    nameService =
     org.omg.CosNaming.NamingContextHelper.narrow (nameServiceObj);
    if (nameService == null)
    {
     System.out.println("nameService = null");
     return;
    }
    // resolve the db object reference
    NameComponent[] IotsName = {
    new NameComponent("IotsMed", "") };
    myDispenser = Iots.IotsDispenserHelper.narrow
          (nameService.resolve(IotsName));
    if (myDispenser == null)
    {
     System.out.println("Failed to resolve IotsMed");
     return;
    }
    myIotsMed  = myDispenser.reserveObject();
   } catch(Exception e)
   { System.out.println("Exception" + e);
   }
 }//intializeORB
 public void releaseDBObject()
 {  try {
     myDispenser.releaseObject(myIotsMed);
    }
    catch (Exception ex)
    {
     System.err.println(ex);
    }
 }//releaseDBObject
}//IotsLSMsBean
```

D. IOTSDISPENSERIMPL

```
// IotsDispenserImpl.java
public class IotsDispenserImpl
    extends Iots._IotsDispenserImplBase
{
```

```
private int maxObjects = 10;
private int numObjects = 0;
private IotsStatus[] IotsMed =
   new IotsStatus[maxObjects];
public IotsDispenserImpl()
{
 super();
}
public IotsDispenserImpl(java.lang.String[] args,
            java.lang.String name, int num)
{
 super(name);
 try
 {
  // get reference to orb
  org.omg.CORBA.ORB orb =
     org.omg.CORBA.ORB.init(args, null);
  // prestart n IotsMed Objects
  numObjects = num;
  for (int i=0; i < numObjects; i++)
  {
   IotsMed[i] = new IotsStatus();
   IotsMed[i].ref = new IotsMedImpl("IotsMed" + (i+1));
   orb.connect(IotsMed[i].ref);
  }
 } catch(Exception e)
 { System.err.println(e);
 }
}
public Iots.IotsMed reserveObject()
{
 for (int i=0; i < numObjects; i++)
 {
  if (!IotsMed[i].inUse)
    { IotsMed[i].inUse = true;
     System.out.println("ClubMed" + (i+1) +
                  " reserved.");
     return IotsMed[i].ref;
    }
 }
 return null;
}
public void releaseObject(Iots.IotsMed IotsMedObject)
```

```
   {
    for (int i=0; i < numObjects; i++)
    {
     if (IotsMed[i].ref == IotsMedObject)
        { IotsMed[i].inUse = false;
          System.out.println("IotsMed" + (i+1) + " released.");
          return;
        }
    }
    System.out.println("Sorry: Reserved Object not found");
    return;
  }
}
class IotsStatus
{
 IotsMedImpl ref;
 boolean    inUse;
 IotsStatus()
 { ref = null;
   inUse = false;
 }
}
```

E. IOTSMEDIMPL

```
// IotsMedImpl.java
import java.util.*;
public class IotsMedImpl extends Iots._IotsMedImplBase
{
 private IotsDBClass myIotsDB;
 private String instanceName;
 static  int    MaxWarehouses=100;
 public IotsMedImpl(java.lang.String name) {
   super(name);
   try
   {
     myIotsDB = new IotsDBClass();
     myIotsDB.connect("system", "manager");
     System.out.println("IotsMedDB Object " + name + " Created");
     instanceName = name;
   } catch (Exception e)
   { System.out.println("System Exception ");
   }
```

```
 }
 public IotsMedImpl()
 {
  super();
 }
 public  void addCustomer(Iots.customerStruct cusStruct)
 {
  try
  {
    myIotsDB.addCustomer(cusStruct);
  } catch (Exception e)
  {
   System.err.println("error in addCustomer() IotsMedImpl");
  }
 }//addCustomer
 public  void modCustomer(Iots.customerStruct cusStruct)
 {
  try
  {
    myIotsDB.modCustomer(cusStruct);
  } catch (Exception e)
  {
   System.err.println("error in modCustomer() IotsMedImpl");
  }
 }//modCustomer
 public  Iots.customerStruct getCustomer(String cusID)
 {
  Iots.customerStruct cusStruct =null;
  try
  {
   cusStruct = myIotsDB.getCustomer(cusID);
  } catch (Exception e)
  {
   System.err.println("error in getCustomer() IotsMedImpl");
  }
  return (cusStruct);
 }// getCustomer
//-------------------------------------------------------------
 public  void addWarehouse(Iots.warehouseStruct wareStruct)
 {
  try
  {
    myIotsDB.addWarehouse(wareStruct);
```

```
    } catch (Exception e)
    {
     System.err.println("error in addWarehouse() IotsMedImpl");
    }
  }//addWarehouse
  public  void modWarehouse(Iots.warehouseStruct wareStruct)
  {
   try
   {
    myIotsDB.modWarehouse(wareStruct);
   } catch (Exception e)
   {
    System.err.println("error in modWarehouse() IotsMedImpl");
   }
  }//modCustomer
  public  Iots.warehouseStruct getWarehouse(String wareID)
  {
   Iots.warehouseStruct wareStruct =null;
   try
   {
    wareStruct = myIotsDB.getWarehouse(wareID);
   } catch (Exception e)
   {
    System.err.println("error in getWarehouse() IotsMedImpl");
   }
   return (wareStruct);
  }// getWarehouse
//-------------------------------------------------------------
  public  Iots.balanceStruct getBalance(String itemID, String wareID)
  {
   Iots.balanceStruct balStruct =null;
   try
   {
    balStruct = myIotsDB.getBalance(itemID, wareID);
   } catch (Exception e)
   {
    System.err.println("error in getBalance() IotsMedImpl");
   }
   return (balStruct);
  }// getBalance
//-------------------------------------------------------------
  public Iots.IDsStruct[] getWareIDs()
  {
```

```
    try
    {
     Iots.IDsStruct[] wareIDs;
     Vector tempV = myIotsDB.getWareIDs();
     wareIDs = new Iots.IDsStruct[tempV.size()];
     tempV.copyInto(wareIDs);
     return (wareIDs);
    } catch (Exception e)
    {
     System.err.println("error in getWareIDS() IotsMedImpl");
     return (null);
    }
  }//getWareIDs
//-----------------------------------------------------------------
  public Iots.ItemIDsStruct[] getItemIDs()
  {
    try
    {
     Iots.ItemIDsStruct[] itemIDs;
     Vector tempV = myIotsDB.getItemIDs();
     itemIDs = new Iots.ItemIDsStruct[tempV.size()];
     tempV.copyInto(itemIDs);
     return (itemIDs);
    } catch (Exception e)
    {
     System.err.println("error in getItemIDs() IotsMedImpl");
     return (null);
    }
  }//getItemIDs
//-----------------------------------------------------------------
  public Iots.ItemIDsStruct2[] getItemIDs2()
  {
    try
    {
     Iots.ItemIDsStruct2[] itemIDs2;
     Vector tempV = myIotsDB.getItemIDs2();
     itemIDs2 = new Iots.ItemIDsStruct2[tempV.size()];
     tempV.copyInto(itemIDs2);
     return (itemIDs2);
    } catch (Exception e)
    {
     System.err.println("error in getItemIDs() IotsMedImpl");
     return (null);
```

```
    }
  }//getItemIDs2
//---------------------------------------------------------------
  public Iots.TransactionStruct[] getTransIDs()
  {
   try
   {
    Iots.TransactionStruct[] transIDs;
    Vector tempV = myIotsDB.getTransIDs();
    transIDs = new Iots.TransactionStruct[tempV.size()];
    tempV.copyInto(transIDs);
    return (transIDs);
   } catch (Exception e)
   {
    System.err.println("error in getTransIDs() IotsMedImpl");
    return (null);
   }
  }//getTransIDs
//---------------------------------------------------------------
  public Iots.CustomerIDsStruct[] getCusIDs()
  {
   try
   {
    Iots.CustomerIDsStruct[] cusIDs;
    Vector tempV = myIotsDB.getCusIDs();
    cusIDs = new Iots.CustomerIDsStruct[tempV.size()];
    tempV.copyInto(cusIDs);
    return (cusIDs);
   } catch (Exception e)
   {
    System.err.println("error in getCusIDs() IotsMedImpl");
    return (null);
   }
  }//getCusIDs
//---------------------------------------------------------------
  public boolean connect(
   java.lang.String user,
   java.lang.String password)
  {
   try
   {
    myIotsDB.connect(user, password);
    return (true);
```

```
    } catch (Exception e)
    {
     return (false);
    }
   }//connect
//-------------------------------------------------------------
  public Iots.OrderResultStruct placeOrder(
              Iots.OrderStruct[] inOrder, Iots.StatStruct stat )
  {
   Iots.OrderResultStruct ordRes=null;
   try
   {
    ordRes = myIotsDB.placeOrder(inOrder, stat);
    return(ordRes);
   } catch (Exception e)
   {
    System.err.println("error in placeOrder() IotsMedImpl");
    System.out.println("EID :"+ordRes.sOrderID);
    System.out.println("Emsg:"+ordRes.sMessage);
    return (new Iots.OrderResultStruct("not av","Un Successfull/MedImp"));
   }
  }// placeOrder
//-------------------------------------------------------------
 public String getDate() {
  Date dt = new Date();
  return(dt.toLocaleString());
 }//getDate
//-------------------------------------------------------------
  public Iots.OrderMainInfoStruct[] getOrderMainInfo(
    String orderID,
    String lDate,
    String hDate)
  {
   try
   {
    Iots.OrderMainInfoStruct[] orders;
    Vector tempV = myIotsDB.getOrderMainInfo(orderID,lDate,hDate);
    orders = new Iots.OrderMainInfoStruct[tempV.size()];
    tempV.copyInto(orders);
    return (orders);
   } catch (Exception e)
   {
    System.err.println("error in getMainInfoStruct() IotsMedImpl");
```

```
    return (null);
    }
  }//getOrderMainInfo
//-----------------------------------------------------------------
  public Iots.StatusStruct[] getStatus(String orderID)
  {
   try
   {
    Iots.StatusStruct[] states;
    Vector tempV = myIotsDB.getStatus(orderID);
    states = new   Iots.StatusStruct[tempV.size()];
    tempV.copyInto(states);
    return (states);
   } catch (Exception e)
   {
    System.err.println("error in getStatus() IotsMedImpl");
    return (null);
   }
  }//getStatus
//-----------------------------------------------------------------
}//IotsMedImpl
```

F. IOTSDBCLASS

```
// IotsDBClass.java
import java.net.URL;
import java.util.*;
import java.sql.*;
public class IotsDBClass {
  Connection con;
  Driver     driver=null;
  ResultSet  resultSet;
  Statement  stmt;
  CallableStatement cstmt;
  PreparedStatement pstmt;
  private   boolean transStatus;
  public IotsDBClass() {
    transStatus = true;
  }
  public boolean connect (String userName, String password)
                  throws Exception
  {
```

```
 boolean connectStatus = false;
 String url ="jdbc:oracle:thin:" + userName +"/" +password +
          "@computerb:1521:oracle";
 try
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  System.out.println("Connecting to ORACLE database");
  con = DriverManager.getConnection(url);
  connectStatus = true;
 } catch (Exception e)
 { System.out.println("System Exception in connect");
  System.err.println(e);
  throw e;
 }
 return connectStatus;
}//connect
public void addCustomer( Iots.customerStruct cusStruct)
{
 try {
   stmt = con.createStatement();
   String addsql =
         "insert into customer " +
         "values('"+ cusStruct.customerID  +"','"+
                  cusStruct.name       +"','"+
                  cusStruct.bStreet    +"','"+
                  cusStruct.bCity      +"','"+
                  cusStruct.bState     +"','"+
                  cusStruct.bZipCode   +"','"+
                  cusStruct.sStreet    +"','"+
                  cusStruct.sCity      +"','"+
                  cusStruct.sState     +"','"+
                  cusStruct.sZipCode   +"','"+
                  cusStruct.telephone  +"','"+
                  cusStruct.phax       +"','"+
                  cusStruct.email      +"','"+
                  cusStruct.creditCardNo +"','"+
                  cusStruct.expDate    +"','"+
                  cusStruct.creditCardType +"')" ;
 stmt.executeUpdate (addsql);
 } catch (Exception e)
 {
  System.err.println("Error in IotsDBClass addcustomer()"+e);
 }
```

```
}//addCustomer
public void modCustomer( Iots.customerStruct cusStruct)
{
 try {
   stmt = con.createStatement();
   String modsql = "update customer " +
    "set CustomerName    = '" + cusStruct.name    +"',"+
    "   ResAddrStreet   = '" + cusStruct.bStreet    +"',"+
    "   ResAddrCity     = '" + cusStruct.bCity      +"',"+
    "   ResAddrState    = '" + cusStruct.bState     +"',"+
    "   ResAddrZip      = '" + cusStruct.bZipCode   +"',"+
    "   ShipAddrStreet  = '" + cusStruct.sStreet    +"',"+
    "   ShipAddrCity    = '" + cusStruct.sCity      +"',"+
    "   ShipAddrState   = '" + cusStruct.sState     +"',"+
    "   ShipAddrZip     = '" + cusStruct.sZipCode   +"',"+
    "   TelephoneNumber = '" + cusStruct.telephone +"',"+
    "   FaxNumber       = '" + cusStruct.phax       +"',"+
    "   EmailAddress    = '" + cusStruct.email      +"',"+
    "   CreditCardNumber = '" + cusStruct.creditCardNo +"',"+
    "   CreditCardExpDate= '" + cusStruct.expDate     +"',"+
    "   CreditCardType  = '" + cusStruct.creditCardType+"' "+
    " where CustomerID   = '" + cusStruct.customerID   +"'";
 stmt.executeUpdate (modsql);
 } catch (Exception e)
 {
  System.err.println("Error in IotsDBClass modcustomer()"+e);
 }
}//modCustomer
public Iots.customerStruct getCustomer( String cusID)
{
 Iots.customerStruct cusStruct = null;
 try {
   stmt = con.createStatement();
   String getsql = "select * from customer " +
              "where CustomerID = '" + cusID +"'";
   ResultSet res = stmt.executeQuery(getsql);
   res.next();
   cusStruct    = new Iots.customerStruct
               (res.getString(1),  res.getString(2),
                res.getString(3),  res.getString(4),
                res.getString(5),  res.getString(6),
                res.getString(7),  res.getString(8),
                res.getString(9),  res.getString(10),
```

```
                    res.getString(11), res.getString(12),
                    res.getString(13), res.getString(14),
                    res.getString(15), res.getString(16)
                   );
    } catch (Exception e)
    {
     System.err.println("Error in IotsDBClass getCustomer()"+e);
    }
   return(cusStruct);
  }//getCustomer
//-------------------------------------------------------------
  public void addWarehouse( Iots.warehouseStruct wareStruct)
  {
   try {
     stmt = con.createStatement();
     String addsql =
           "insert into WAREHOUSE " +
           "values('"+ wareStruct.sWarehouseID     +"','"+
                    wareStruct.sWarehouseName   +"','"+
                    wareStruct.sWareAddrStreet  +"','"+
                    wareStruct.sWareAddrCity    +"','"+
                    wareStruct.sWareAddrState   +"','"+
                    wareStruct.sWareAddrZip     +"','"+
                    wareStruct.sFaxNumber       +"','"+
                    wareStruct.sTelephoneNumber +"')" ;
    stmt.executeUpdate (addsql);
    } catch (Exception e)
    {
     System.err.println("Error in IotsDBClass addWarehouse()"+e);
    }
  }//addWarehouse
  public void modWarehouse( Iots.warehouseStruct wareStruct)
  {
   try {
     stmt = con.createStatement();
     String modsql = "update WAREHOUSE " +
       "set WarehouseName  = '" + wareStruct.sWarehouseName  +"','"+
       "    WareAddrStreet = '" + wareStruct.sWareAddrStreet      +"','"+
       "    WareAddrCity   = '" + wareStruct.sWareAddrCity        +"','"+
       "    WareAddrState  = '" + wareStruct.sWareAddrState       +"','"+
       "    WareAddrZip    = '" + wareStruct.sWareAddrZip         +"','"+
       "    FaxNumber      = '" + wareStruct.sFaxNumber           +"','"+
       "    TelephoneNumber= '" + wareStruct.sTelephoneNumber+"'"+
```

```
      " where WarehouseID = '" + wareStruct.sWarehouseID      +"'";
   stmt.executeUpdate (modsql);
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass modWarehouse()"+e);
   }
 }//modWarehouse
 public Iots.warehouseStruct getWarehouse( String wareID)
 {
 Iots.warehouseStruct wareStruct = null;
 try {
    stmt = con.createStatement();
    String getsql = "select * from WAREHOUSE " +
                "where WarehouseID = '" + wareID +"'";
    ResultSet res = stmt.executeQuery(getsql);
    res.next();
    wareStruct     = new Iots.warehouseStruct
                (res.getString(1),  res.getString(2),
                 res.getString(3),  res.getString(4),
                 res.getString(5),  res.getString(6),
                 res.getString(7),  res.getString(8)
                );
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass getWarehouse()"+e);
   }
   return(wareStruct);
 }//getWarehouse
//-------------------------------------------------------------
 public Iots.balanceStruct getBalance( String itemID, String wareID)
 {
 Iots.balanceStruct balStruct = null;
 try {
  stmt = con.createStatement();
  String getsql = "select ItemID, ItemDescreption, PartNumber, " +
  " TaxableItem, SalePrice, UnitOfSale, WarehouseID,WarehouseName, "+
  " Balance, ReorderLevel, ReorderQty " +
  " from item, inventory, warehouse " +
  " where ItemID = '" + itemID +"' AND ItemID = ItemID_FK AND " +
  "   WarehouseID= '" + wareID +"' AND WarehouseID=WarehouseID_FK";

  ResultSet res= stmt.executeQuery(getsql);
  res.next();
```

```
    balStruct    = new Iots.balanceStruct
                  (res.getString(1),  res.getString(2),
                   res.getString(3),  res.getString(4),
                   res.getFloat(5),  res.getString(6),
                   res.getString(7),  res.getString(8),
                   res.getFloat(9),  res.getFloat(10),
                   res.getFloat(11)
                  );
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getBalance()");
     while (ex !=null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message : " +ex.getMessage());
        System.out.println("VendorCo: " +ex.getErrorCode());
        ex.printStackTrace(System.out);
        ex= ex.getNextException();
        System.out.println("_____");
     }
    }
    return(balStruct);
  }//getBalance
//---------------------------------------------------------------
  public Vector getWareIDs()
  {
   try {
    Vector wareIDs = new Vector();
    stmt = con.createStatement();
    String getsql =
      "select WarehouseID from Warehouse group by WarehouseID";
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
     wareIDs.addElement( new Iots.IDsStruct(res.getString(1)) );
    }//while
    return(wareIDs);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getWareIDs()");
     while (ex !=null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message : " +ex.getMessage());
        System.out.println("VendorCo: " +ex.getErrorCode());
        ex.printStackTrace(System.out);
```

```
        ex= ex.getNextException();
        System.out.println("_____");
      }
     return(null);
    }
  }//getWareIDs
//-------------------------------------------------------------
  public Vector getItemIDs()
  {
   try {
    Vector itemIDs = new Vector();
    stmt = con.createStatement();
    String getsql =
      "select ItemID from ITEM group by ItemID";
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
     itemIDs.addElement( new Iots.ItemIDsStruct(res.getString(1)) );
    }//while
    return(itemIDs);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getItemIDs()");
     while (ex !=null) {
        System.out.println("SQLState: " + ex.getSQLState());
        System.out.println("Message : " +ex.getMessage());
        System.out.println("VendorCo: " +ex.getErrorCode());
        ex.printStackTrace(System.out);
        ex= ex.getNextException();
        System.out.println("_____");
      }
     return(null);
    }
  }//getItemIDs
//-------------------------------------------------------------
  public Vector getItemIDs2()
  {
   try {
    Vector itemIDs2 = new Vector();
    stmt = con.createStatement();
    String getsql =
      "select ItemID, ItemDescreption, SalePrice from ITEM";
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
```

```
    itemIDs2.addElement( new Iots.ItemIDsStruct2
        (res.getString(1),res.getString(2), res.getFloat(3) ));
    }//while
    return(itemIDs2);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getItemIDs2()");
     while (ex !=null) {
        System.out.println("Message : " +ex.getMessage());
        ex= ex.getNextException();
     }
    return(null);
    }
  }//getItemIDs2
//-------------------------------------------------------------
 public Vector getTransIDs()
  {
   try {
    Vector transIDs = new Vector();
    stmt = con.createStatement();
    String getsql =
     "select * from TRANSACTION";
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
     transIDs.addElement( new Iots.TransactionStruct(
                res.getString(1),res.getString(2) ));
    }//while
    return(transIDs);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getTransIDs()");
     while (ex !=null) {
        System.out.println("Message : " +ex.getMessage());
        ex.printStackTrace(System.out);
        ex= ex.getNextException();
     }
    return(null);
    }
  }//getTransIDs
//-------------------------------------------------------------
 public Vector getCusIDs()
  {
   try {
```

```
    Vector cusIDs = new Vector();
    stmt = con.createStatement();
    String getsql =
      "select CustomerID, CustomerName from CUSTOMER";
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
     cusIDs.addElement( new Iots.CustomerIDsStruct(
                  res.getString(1),res.getString(2) ));
    }//while
    return(cusIDs);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getCusIDs()");
     while (ex !=null) {
        System.out.println("Message : " +ex.getMessage());
        ex.printStackTrace(System.out);
        ex= ex.getNextException();
     }
    return(null);
    }
  }//getCusIDs
//-------------------------------------------------------------
  public Iots.OrderResultStruct placeOrder(
            Iots.OrderStruct[] inOrder,Iots.StatStruct stat )
  {
   Connection wcon = con;
   transStatus= true;
   try {
       wcon.setAutoCommit(false);
       int invoiceID = getInvoiceID(wcon);
       int orderID   = getOrderID(wcon);
       openInvoice(wcon,
                java.lang.String.valueOf(invoiceID),
                stat.sOrderDateAndTime,
                stat.sTotalDue
                );
       openInvoiceDetails(wcon,
                java.lang.String.valueOf(invoiceID),
                java.lang.String.valueOf(orderID)
                );
       openOrder(wcon,
               java.lang.String.valueOf(orderID),
               stat.sCustomerID,
```

```
                stat.sOrderDateAndTime,
                stat.sTransactionCode,
                stat.sTaxAll,
                stat.sTotalDue,
                java.lang.String.valueOf(invoiceID)
                );
        for ( int i=0; i< inOrder.length; i++)
        {
      updatOrderDetails(wcon,
                java.lang.String.valueOf(orderID),
                inOrder[i].itemID,
                inOrder[i].quantity,
                inOrder[i].salePrice,
                inOrder[i].extendedPrice
                );
        }//for
  updateOrderStatus(wcon,
                java.lang.String.valueOf(orderID),
                stat.sOrderDateAndTime,
                stat.sUserID,
                "NEW",
                "Order is placed recently"
                );
   setInvoiceID(wcon);
   setOrderID(wcon);
   if (transStatus) //all transactions completed with no errors
       wcon.commit();
   else wcon.rollback();
   return(new Iots.OrderResultStruct(
        java.lang.String.valueOf(orderID),
        "Please write down the order's ID for tracking purposes"));
   } catch (Exception e)
   {
   System.err.println("Error in IotsDBClass placeOrder()"+e);
   return(new Iots.OrderResultStruct("no id av","UnSuccessful/DB"));
   }
 }//placeOrder
//-------------------------------------------------------------
public void updateOrderStatus(
  Connection wcon,
  String    orderID,
  String    sOrderDateAndTime,
  String    sUserID,
```

```
 String    status,
 String    remarks)

{try
 {
  stmt = wcon.createStatement();
  String dd   = sOrderDateAndTime.substring(4,6);
  String mmm  = sOrderDateAndTime.substring(0,3);
  String yy   = sOrderDateAndTime.substring(10,12);
  String wdate= dd + "-" + mmm + "-" + yy;
  String getsql = "insert into ORDER_STATUS   " +
      "values ('" + orderID        +"',TO_DATE('" +
                wdate          +"'),'" +
                sUserID         +"','"  +
                status         +"','" +
                remarks         +"')";
  System.out.println(getsql);
  stmt.executeUpdate(getsql);
  } catch (Exception e)
  {
   System.err.println("Error in IotsDBClass updateOrderStatus()"+e);
   transStatus=false;
  }
}//updateOrderStatus
//-----------------------------------------------------------------
public void updatOrderDetails(
     Connection wcon,
     String orderID,
     String itemID,
     int   quantity,
     float  salePrice,
     float  extendedPrice)
{try
 {
  stmt = wcon.createStatement();
  String getsql = "insert into ORDER_DETAILS   " +
      "values ('" +
                orderID        +"','" +
                itemID         +"'," +
                quantity       +"," +
                salePrice      +"," +
                extendedPrice    +")";
  System.out.println(getsql);
```

```
   stmt.executeUpdate(getsql);
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass updateOrderDetails()"+e);
    transStatus=false;
   }
 }//updateOrderDetails
//-----------------------------------------------------------------
public void openOrder(
   Connection wcon,
   String    orderID,
   String    sCustomerID,
   String    sOrderDateAndTime,
   String    sTransactionCode,
   float     sTaxAll,
   float     sTotalDue,
   String    invoiceID)
 {try
  {
   stmt = wcon.createStatement();
   String dd   = sOrderDateAndTime.substring(4,6);
   String mmm  = sOrderDateAndTime.substring(0,3);
   String yy   = sOrderDateAndTime.substring(10,12);
   String wdate= dd + "-" + mmm + "-" + yy;
   String getsql = "insert into ORDERS   " +
       "values (" +
                 orderID       +"'," +
                sCustomerID +"',TO_DATE(" +
                wdate          +"),'" +
                sTransactionCode  +"'," +
                sTaxAll        +"," +
                sTotalDue      +",'"+
                invoiceID      +"')";
   System.out.println(getsql);
   stmt.executeUpdate(getsql);
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass openOrder()"+e);
    transStatus=false;
   }
 }//openOrder
//-----------------------------------------------------------------
 public void openInvoiceDetails(Connection wcon, String invoiceID,
```

```
                String orderID)
 {try
  {
    stmt = wcon.createStatement();
    String getsql = "insert into INVOICE_DETAILS  " +
       "values ('" + invoiceID +"','" + orderID +"')";
    System.out.println(getsql);
    stmt.executeUpdate(getsql);
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass OpenInvoiceDetails()"+e);
    transStatus=false;
   }
 }//openInvoiceDetails
//---------------------------
 public void openInvoice(Connection wcon, String invoiceID,
                String sOrderDateAndTime, float sTotalDue)
 {try
  {
    stmt = wcon.createStatement();
    String dd   = sOrderDateAndTime.substring(4,6);
    String mmm  = sOrderDateAndTime.substring(0,3);
    String yy   = sOrderDateAndTime.substring(10,12);
    String wdate= dd + "-" + mmm + "-" + yy;
    String getsql = "insert into INVOICE " +
       "values ('" + invoiceID +"',TO_DATE('" + wdate +
       "')," + sTotalDue +")";
    System.out.println(getsql);
    stmt.executeUpdate(getsql);
   } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass OpenInvoiceID()"+e);
    transStatus=false;
   }
 }//openInvoice
//-------------------------------------------------------------
 public int getInvoiceID(Connection wcon)
 {try
  {
    stmt = wcon.createStatement();
    String getsql = "select IDs from invoicesIDs where ikey =1";
    ResultSet res = stmt.executeQuery(getsql);
    res.next();
```

```
      return (res.getInt(1));
     } catch (Exception e)
     {
      System.err.println("Error in IotsDBClass getInvoiceID()"+e);
      transStatus=false;
      return(0);
     }
  }//getInvoiceID
//---------------------------
  public void setInvoiceID(Connection wcon)
  {try
    {
      stmt = wcon.createStatement();
      String getsql = "update invoicesIDs set IDs = IDs + 1" +
                  " where ikey=1";
      stmt.executeUpdate(getsql);
    } catch (Exception e)
     {
      System.err.println("Error in IotsDBClass setInvoiceID()"+e);
      transStatus=false;
     }
  }//setInvoiceID
//---------------------------
  public int getOrderID(Connection wcon)
  {try
    {
      stmt = wcon.createStatement();
      String getsql = "select IDs from ordersIDs where okey =1";
      ResultSet res = stmt.executeQuery(getsql);
      res.next();
      return (res.getInt(1));
     } catch (Exception e)
     {
      System.err.println("Error in IotsDBClass getOrderID()"+e);
      transStatus=false;
      return(0);
     }
  }//getOrderID
//---------------------------
  public void setOrderID(Connection wcon)
  {try
    {
      stmt = wcon.createStatement();
```

```
    String getsql = "update ordersIDs set IDs = IDs + 1" +
                    " where okey=1";
    stmt.executeUpdate(getsql);
    } catch (Exception e)
   {
    System.err.println("Error in IotsDBClass setOrderID()"+e);
    transStatus=false;
   }
 }//setOrderID
//-------------------------------------------------------------
 public Vector getOrderMainInfo(
  java.lang.String orderID,
  java.lang.String lDate,
  java.lang.String hDate)
 {
 String getsql;
 try {
  Vector ordMain = new Vector();
  stmt = con.createStatement();
  if (orderID.compareTo("")==0) // orderID= spaces
  {
   getsql = "select * from ORDERS where OrderDateAndTime >= " +
          "TO_Date('" + lDate + "') AND OrderDateAndTime <= " +
          "To_DATE('" + hDate + "')";
  }
  else
  {
   getsql ="select * from ORDERS where OrderID = '" +orderID+"'";
  }
  System.out.println(getsql);
  ResultSet res= stmt.executeQuery(getsql);
  while (res.next()) { ordMain.addElement( new Iots.OrderMainInfoStruct(
      res.getString(1),res.getString(2), res.getString(5),
      res.getString(6),res.getString(3), res.getString(7),
      getLastStatus(res.getString(1)) ));
  }//while
  return(ordMain);
  } catch (SQLException ex)
  {
   System.err.println("Error in IotsDBClass getOrderMainInfo()");
   while (ex !=null) {
      System.out.println("Message : " +ex);
      ex= ex.getNextException();
```

```
      }
     return(null);
     }
   }//getOrderMainInfo
//-----------------------------------------------
   public String getLastStatus(java.lang.String orderID)
   {
    String str =null;
   try {
    String getsql;
    stmt = con.createStatement();
    getsql =" SELECT Status" +
          " from ORDER_STATUS where OrderID_FK = '" +orderID+"'";
    System.out.println(getsql);
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) {
       str = res.getString(1);
    }//while
    return(str);
    } catch (SQLException ex)
    {
     System.err.println("Error in IotsDBClass getLastStatus()");
     while (ex !=null) {
        System.out.println("Message : " +ex);
        ex= ex.getNextException();
     }
    return(new String("Not Found"));
    }
   }//getLastStatus
//---------------------------------------------------------------
   public Vector getStatus(java.lang.String orderID)
   {
   try {
    String getsql;
    Vector ordstat = new Vector();
    stmt = con.createStatement();
    getsql ="select StatusDateAndTime, Status, Remarks " +
          " from ORDER_STATUS where OrderID_FK = '" +orderID+"'";
    System.out.println(getsql);
    ResultSet res= stmt.executeQuery(getsql);
    while (res.next()) { ordstat.addElement(
        new Iots.StatusStruct(
        res.getString(1),res.getString(2), res.getString(3) ));
```

```
   }//while
   return(ordstat);
   } catch (SQLException ex)
   {
    System.err.println("Error in IotsDBClass getStatus()");
    while (ex !=null) {
       System.out.println("Message : " +ex);
       ex= ex.getNextException();
    }
   return(null);
   }
  }//getStatus
}//IotsDBClass
```

G. SECURITYBEAN

```
//SecurityBean.java
import java.awt.*;
import java.awt.Graphics;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class SecurityBean extends JPanel {
 private Iots.IotsDispenser  myDispenser;
 private JPanel     pl;
 private Graphics  gContext;
 private JLabel     userL, passwordL,dummyL;
 private JTextField user, status;
 private JPasswordField password;
 private JButton    logOnb;
 private GridBagLayout       gbLayout;
 private GridBagConstraints   gbConstraints;
 private Iots.IotsMed        myIotsMed;
 public SecurityBean(Iots.IotsMed inMed,
       Iots.IotsDispenser  myDisp) {
  myIotsMed= inMed;
  myDispenser= myDisp;
  gbLayout = new GridBagLayout();
  setLayout(gbLayout);
```

```
gbConstraints = new GridBagConstraints();
dummyL     = new JLabel (" ");
userL      = new JLabel ("User ID");
passwordL  = new JLabel ("Password");
user       = new JTextField("",15);
password   = new JPasswordField("",15);
status     = new JTextField("",30);
status.setEditable(false);
logOnb     = new JButton("Logon");
logOnb.addActionListener(new ActionListener()
     {
      public void actionPerformed(ActionEvent event) {
       try {
        if (myIotsMed.connect(user.getText(),
             password.getText()))
        {
            status.setText("Access is Autherized for user=" +
               user.getText());
        }
        else {
            status.setText("NOT Autherized. Check" +
               " UserID and/or Pasword are valid");
            //terminate the connection
            myDispenser.releaseObject(myIotsMed);
        }
       }
       catch (Exception ex) {
        status.setText("Exception in SecurityBean " +ex);
         }
          }
    } );
gbConstraints.fill = GridBagConstraints.NONE;
gbConstraints.anchor=GridBagConstraints.NORTHWEST;
addComponent(dummyL,0,0,1,1);
addComponent(dummyL,1,0,1,1);
addComponent(dummyL,2,0,1,1);
addComponent(userL, 3,0,1,1);
addComponent(user , 3,1,1,1);
addComponent(passwordL,4,0,1,1);
addComponent(password ,4,1,1,1);
gbConstraints.fill = GridBagConstraints.NONE;
addComponent(logOnb   ,5,0,2,1);
addComponent(status   ,6,0,2,1);
```

```
  setVisible(true);
 }// constructor
 private void addComponent( Component c, int row,
         int column, int width, int height) {
  gbConstraints.gridx = column;
  gbConstraints.gridy = row;
  gbConstraints.gridwidth= width;
  gbConstraints.gridheight= height;
  gbLayout.setConstraints(c, gbConstraints);
  add(c);
 }//addComponent
}//SecurityBean
```

H. WAREHOUSEINFOBEAN

```
//WarehouseInfoBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
public class WarehouseInfoBean extends JPanel {
 private JPanel pl;
 private JLabel warehouseIDL, warehouseNameL,
            wareAddrL, wareAddrStreetL,
            wareAddrCityL, wareAddrStateL, wareAddrZipL,
            seperatorL,faxNumberL,telephoneNumberL;
 private JTextField
            warehouseID, warehouseName, wareAddrStreet,
            wareAddrCity, wareAddrZip,
            seperator,faxNumber,telephoneNumber, status;
 private String [] states =
            {" ","AL","AK","AR","AZ","CA","CO","CT","DE","DC","FL",
                  "GA","HI","ID","IL","IN","IA","KS","KY","LA","ME",
             "MD","MA","MI","MN","MS","MO","MT","NE","NV","NH",
             "NJ","NM","NY","NC","ND","OH","OK","OR","PA","RI",
             "SC","SD","TN","TX","UT","VT","VA","WA","WV","WI",
             "WY"};
 private JComboBox wareAddrState;
 private JButton          addWareb, getWareb, updateWareb;
 private GridBagLayout    gbLayout;
```

```
private GridBagConstraints   gbConstraints;
private Iots.warehouseStruct wareStruct;
private Iots.IotsMed         myIotsMed;
public WarehouseInfoBean(Iots.IotsMed inMed) {
 myIotsMed= inMed;
 gbLayout = new GridBagLayout();
 setLayout(gbLayout);
 gbConstraints = new GridBagConstraints();
 warehouseIDL    = new JLabel ("Warehouse ID");
 warehouseNameL  = new JLabel ("Name");
 wareAddrL       = new JLabel ("Address");
 wareAddrL.setForeground(java.awt.Color.red);
 wareAddrStreetL = new JLabel("Street");
 wareAddrCityL   = new JLabel("City");
 wareAddrStateL  = new JLabel("State");
 wareAddrZipL    = new JLabel("Zip Code");
 seperatorL      = new JLabel("_____________");
 faxNumberL      = new JLabel("Fax");
 telephoneNumberL= new JLabel("Telephone");
 warehouseID     = new JTextField("",5);
 warehouseName   = new JTextField("",25);
 wareAddrStreet  = new JTextField("",25);
 wareAddrCity    = new JTextField("",15);
 wareAddrState   = new JComboBox();
 for (int i=0; i< states.length; i++) {
   wareAddrState.addItem(states[i]);
 }//for
 wareAddrZip     = new JTextField("",10);
 faxNumber       = new JTextField("",12);
 telephoneNumber = new JTextField("",12);
 status          = new JTextField("",40);
 addWareb        = new JButton("Add Warehouse");
 addWareb.setForeground(java.awt.Color.red);
 getWareb        = new JButton("Get Warehouse");
 getWareb.setForeground(java.awt.Color.red);
 updateWareb     = new JButton("Mod Warehouse");
 updateWareb.setForeground(java.awt.Color.red);
 addWareb.addActionListener(new ActionListener()
      {
       public void actionPerformed(ActionEvent event) {
        try {
                status.setText("");
           wareStruct = new Iots.warehouseStruct(
```

```
            warehouseID.getText(),
            warehouseName.getText(),
            wareAddrStreet.getText(),
                  wareAddrCity.getText(),
            java.lang.String.valueOf(wareAddrState.getSelectedItem()),
                  wareAddrZip.getText(),
            faxNumber.getText(),
            telephoneNumber.getText()
          );
              myIotsMed.addWarehouse(wareStruct);
          }
          catch (Exception ex) {
           status.setText("Exception AddWarehouse: " +ex.toString());
          }
           }
    } );
getWareb.addActionListener(new ActionListener()
    {
     public void actionPerformed(ActionEvent event) {
      try {
              status.setText("");
        wareStruct = myIotsMed.getWarehouse(warehouseID.getText());
          warehouseID.setText(wareStruct.sWarehouseID);
          warehouseName.setText(wareStruct.sWarehouseName);
          wareAddrStreet.setText(wareStruct.sWareAddrStreet);
                wareAddrCity.setText(wareStruct.sWareAddrCity);
          wareAddrState.setSelectedItem(wareStruct.sWareAddrState);
                wareAddrZip.setText(wareStruct.sWareAddrZip);
          faxNumber.setText(wareStruct.sFaxNumber);
          telephoneNumber.setText(wareStruct.sTelephoneNumber);
          }
        catch (Exception ex) {
         status.setText("Exception getWarehouse: " +ex.toString());
        }
         }
    } );
updateWareb.addActionListener(new ActionListener()
    {
     public void actionPerformed(ActionEvent event) {
      try {
              status.setText("");
        wareStruct = new Iots.warehouseStruct(
          warehouseID.getText(),
```

```
            warehouseName.getText(),
            wareAddrStreet.getText(),
                  wareAddrCity.getText(),
            java.lang.String.valueOf(wareAddrState.getSelectedItem()),
                  wareAddrZip.getText(),
            faxNumber.getText(),
            telephoneNumber.getText()
          );
             myIotsMed.modWarehouse(wareStruct);
          }
          catch (Exception ex) {
           status.setText("Exception modWarehouse: " +ex.toString());
          }
           }
      } );
 gbConstraints.fill = GridBagConstraints.NONE;
 gbConstraints.anchor=GridBagConstraints.NORTHWEST;
 addComponent(warehouseIDL,0,0,1,1);
 addComponent(warehouseID, 0,1,1,1);
 addComponent(warehouseNameL,0,2,1,1);
 addComponent(warehouseName,0,3,2,1);
 addComponent(wareAddrL,2,0,1,1);
 addComponent(getWareb,7,3,1,1);
 addComponent(addWareb,8,3,1,1);
 addComponent(updateWareb,9,3,1,1);
 addComponent(wareAddrStreetL,3,0,1,1);
 addComponent(wareAddrStreet, 3,1,3,1);
 addComponent(wareAddrCityL,4,0,1,1);
 addComponent(wareAddrCity,4,1,1,1);
 addComponent(wareAddrStateL,5,0,1,1);
 addComponent(wareAddrState,5,1,1,1);
 addComponent(wareAddrZipL,6,0,1,1);
 addComponent(wareAddrZip,6,1,1,1);
 addComponent(seperatorL,7,0,1,1);
 addComponent(faxNumberL,8,0,1,1);
 addComponent(faxNumber,8,1,1,1);
 addComponent(telephoneNumberL,9,0,1,1);
 addComponent(telephoneNumber,9,1,1,1);
 addComponent(status,10,0,6,1);
 setVisible(true);
}// constructor
private void addComponent( Component c, int row, int column, int width,
                  int height) {
```

```
  gbConstraints.gridx = column;
  gbConstraints.gridy = row;
  gbConstraints.gridwidth= width;
  gbConstraints.gridheight= height;
  gbLayout.setConstraints(c, gbConstraints);
  add(c);
 }//WarehouseInfoBean
```

I. CUSTOMERBEAN

```
//IotsLSMsBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
public class CustomerBean extends JPanel {
 private JPanel pl;
 private JLabel customerIDL, nameL, bAddressL, bStreetL,
           bCityL, bStateL, bZipCodeL, sAddressL,
           sStreetL, sCityL, sStateL, sZipCodeL,
           seperator,telephoneL, phaxL, emailL,
           creditCardNoL, expDateL, creditCardTypeL, myL;
 private JTextField
           customerID, name, bStreet, bCity, bZipCode, sStreet,
           sCity, sZipCode, telephone, phax, email, creditCardNo,
           status;
 private String [] states =
           {" ","AL","AK","AR","AZ","CA","CO","CT","DE","DC","FL",
                 "GA","HI","ID","IL","IN","IA","KS","KY","LA","ME",
            "MD","MA","MI","MN","MS","MO","MT","NE","NV","NH",
            "NJ","NM","NY","NC","ND","OH","OK","OR","PA","RI",
            "SC","SD","TN","TX","UT","VT","VA","WA","WV","WI",
            "WY"};
 private String [] months = {"   ","JAN","FEB","MAR","APR","MAY","JUN",
                   "JUL","AUG","SEP","OCT","NOV","DEC"};
 private JComboBox bState,sState,expMM,expYY,creditCardType;
 private JPanel    datePanel;
 private JButton           addCustomerb, getCustomerb, updateCustomerb;
 private GridBagLayout     gbLayout;
 private GridBagConstraints  gbConstraints;
```

```
private Iots.customerStruct cusStruct;
private Iots.IotsMed        myIotsMed;
public CustomerBean(Iots.IotsMed inMed) {
 myIotsMed= inMed;
 gbLayout = new GridBagLayout();
 setLayout(gbLayout);
 gbConstraints = new GridBagConstraints();
 customerIDL    = new JLabel ("Customer ID");
 nameL          = new JLabel ("Name");
 bAddressL      = new JLabel("Billing Address");
 bAddressL.setForeground(java.awt.Color.red);
 bStreetL       = new JLabel("Street");
 bCityL         = new JLabel("City");
 bStateL        = new JLabel("State");
 bZipCodeL      = new JLabel("Zip Code");
 sAddressL      = new JLabel("Shipping Address");
 sAddressL.setForeground(java.awt.Color.red);
 sStreetL       = new JLabel("Street");
 sCityL         = new JLabel("City");
 sStateL        = new JLabel("State");
 sZipCodeL      = new JLabel("Zip Code");
 seperator      = new JLabel("_____________");
 telephoneL     = new JLabel("Telephone");
 phaxL          = new JLabel("Fax");
 emailL         = new JLabel("Email");
 creditCardNoL  = new JLabel("Credit Card No");
 expDateL       = new JLabel("Exp Date");
 creditCardTypeL= new JLabel("Credit Card Type");
 myL            = new JLabel("   Month      Year");
 customerID     = new JTextField("",7);
 name           = new JTextField("",15);
 bStreet        = new JTextField("",15);
 bCity          = new JTextField("",15);
 bState         = new JComboBox();
 sState         = new JComboBox();
 for (int i=0; i< states.length; i++) {
   bState.addItem(states[i]);
   sState.addItem(states[i]);
 }//for
 bZipCode       = new JTextField("",10);
 sStreet        = new JTextField("",15);
 sCity          = new JTextField("",15);
 sZipCode       = new JTextField("",10);
```

```
telephone    = new JTextField("",10);
phax         = new JTextField("",10);
email        = new JTextField("",15);
creditCardNo = new JTextField("",16);
datePanel    = new JPanel();
expMM        = new JComboBox();
expYY        = new JComboBox();
for (int i=0; i<months.length; i++)
   { expMM.addItem(months[i]);}
for (int i=00; i<20; i++)
   { expYY.addItem(java.lang.String.valueOf(i));}
datePanel.add(expMM);
datePanel.add(expYY);
creditCardType= new JComboBox();
creditCardType.addItem("Visa");
creditCardType.addItem("MasterCard");
creditCardType.addItem("Discovery");
status       = new JTextField("",40);
addCustomerb = new JButton("Add Customer");
addCustomerb.setForeground(java.awt.Color.red);
getCustomerb = new JButton("Get Customer");
getCustomerb.setForeground(java.awt.Color.red);
updateCustomerb = new JButton("Mod Customer");
updateCustomerb.setForeground(java.awt.Color.red);
addCustomerb.addActionListener(new ActionListener()
     {
      public void actionPerformed(ActionEvent event) {
       try {
          status.setText("");
          cusStruct = new Iots.customerStruct(
                    customerID.getText(),
                    name.getText(),
                    bStreet.getText(),
                    bCity.getText(),
                    java.lang.String.valueOf(bState.getSelectedItem()),
                    bZipCode.getText(),
                    sStreet.getText(),
                    sCity.getText(),
               java.lang.String.valueOf(sState.getSelectedItem()),
                    sZipCode.getText(),
                    telephone.getText(),
                    phax.getText(),
                    email.getText(),
```

```
                creditCardNo.getText(),
                "28-" +
            java.lang.String.valueOf(expMM.getSelectedItem()) +
            "-" +
            java.lang.String.valueOf(expYY.getSelectedItem()),
                java.lang.String.valueOf(creditCardType.getSelectedItem())
        );
            myIotsMed.addCustomer(cusStruct);
        }
        catch (Exception ex) {
         status.setText("Exception AddCustomer: " +ex.toString());
        }
         }
    } );
getCustomerb.addActionListener(new ActionListener()
    {
     public void actionPerformed(ActionEvent event) {
      try {
              status.setText("");
        cusStruct = myIotsMed.getCustomer(customerID.getText());
                customerID.setText(cusStruct.customerID);
                name.setText(cusStruct.name);
                bStreet.setText(cusStruct.bStreet);
                bCity.setText(cusStruct.bCity);
                bState.setSelectedItem(cusStruct.bState);
                bZipCode.setText(cusStruct.bZipCode);
                sStreet.setText(cusStruct.sStreet);
                sCity.setText(cusStruct.sCity);
            sState.setSelectedItem(cusStruct.sState);
                sZipCode.setText(cusStruct.sZipCode);
                telephone.setText(cusStruct.telephone);
                phax.setText(cusStruct.phax);
                email.setText(cusStruct.email);
                creditCardNo.setText(cusStruct.creditCardNo);

            char [] datechar = (cusStruct.expDate).toCharArray();
            String  mm = java.lang.String.copyValueOf(datechar, 5,2);
            int     mmi= java.lang.Integer.parseInt(mm);
            expMM.setSelectedItem(months[mmi]);
            String  yy = java.lang.String.copyValueOf(datechar, 2,2);
            int     yyi= java.lang.Integer.parseInt(yy);
            expYY.setSelectedItem(java.lang.String.valueOf(yyi));
                creditCardType.setSelectedItem(cusStruct.creditCardType);
```

```
        }
        catch (Exception ex) {
         status.setText("Exception getCustomer: " +ex.toString());
        }
         }
    } );
updateCustomerb.addActionListener(new ActionListener()
    {
     public void actionPerformed(ActionEvent event) {
      try {
          status.setText("");
         cusStruct = new Iots.customerStruct(
                customerID.getText(),
                name.getText(),
                bStreet.getText(),
                bCity.getText(),
                java.lang.String.valueOf(bState.getSelectedItem()),
                bZipCode.getText(),
                sStreet.getText(),
                sCity.getText(),
           java.lang.String.valueOf(sState.getSelectedItem()),
                sZipCode.getText(),
                telephone.getText(),
                phax.getText(),
                email.getText(),
           java.lang.String.copyValueOf (
           ((creditCardNo.getText()).toCharArray()),0,16),
                "28-" +
           java.lang.String.valueOf(expMM.getSelectedItem()) +
           "-" +
           java.lang.String.valueOf(expYY.getSelectedItem()),
                java.lang.String.valueOf(creditCardType.getSelectedItem())
        );
           myIotsMed.modCustomer(cusStruct);
        }
        catch (Exception ex) {
         status.setText("Exception modCustomer: " +ex.toString()+cusStruct);
        }
         }
    } );
gbConstraints.fill = GridBagConstraints.NONE;
gbConstraints.anchor=GridBagConstraints.NORTHWEST;
addComponent(customerIDL,0,0,1,1);
```

```
 addComponent(customerID, 0,1,1,1);
 addComponent(getCustomerb,0,2,1,1);
 addComponent(nameL,0,3,1,1);
 addComponent(name,0,4,3,1);
 addComponent(bAddressL,2,0,1,1);
 addComponent(addCustomerb,2,2,1,1);
 addComponent(updateCustomerb,3,2,1,1);
 addComponent(bStreetL,3,0,1,1);
 addComponent(bStreet, 3,1,3,1);
 addComponent(bCityL,4,0,1,1);
 addComponent(bCity,4,1,1,1);
 addComponent(bStateL,5,0,1,1);
 addComponent(bState,5,1,1,1);
 addComponent(bZipCodeL,6,0,1,1);
 addComponent(bZipCode,6,1,1,1);
 addComponent(sAddressL,2,4,1,1);
 addComponent(sStreet, 3,4,3,1);
 addComponent(sCity,4,4,1,1);
 addComponent(sState,5,4,1,1);
 addComponent(sZipCode,6,4,1,1);
 addComponent(seperator,7,0,1,1);
 addComponent(telephoneL,8,0,1,1);
 addComponent(telephone,8,1,1,1);
 addComponent(phaxL,8,2,1,1);
 addComponent(phax,8,4,1,1);
 addComponent(emailL,9,0,1,1);
 addComponent(email,9,1,3,1);
 addComponent(myL,9,3,2,1);
 addComponent(creditCardNoL,10,0,1,1);
 addComponent(creditCardNo,10,1,1,1);
 addComponent(expDateL,10,2,1,1);
 addComponent(datePanel,10,3,2,1);
 addComponent(creditCardTypeL,11,0,1,1);
 addComponent(creditCardType,11,1,1,1);
 addComponent(status,12,0,6,1);
 setVisible(true);
}// constructor
private void addComponent( Component c, int row, int column, int width,
                  int height) {
 gbConstraints.gridx = column;
 gbConstraints.gridy = row;
 gbConstraints.gridwidth= width;
 gbConstraints.gridheight= height;
```

```
  gbLayout.setConstraints(c, gbConstraints);
  add(c);
 }//addComponent
}//AddCustomerBean
```

J. PLACEORDERBEAN

```
//PlaceOrderBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class PlaceOrderBean extends JPanel {
 private JPanel pl;
 private Iots.TransactionStruct[] transIDs;
 private Iots.CustomerIDsStruct[] cusIDs;
 private Iots.ItemIDsStruct2[]    itemIDs2;
 private Iots.OrderResultStruct   ordRes;
 private Iots.StatStruct          stat;
 private String                   userID;
 private JLabel customerIDL, transactionCodeL,dateL,
           itemIDL, itemDescreptionL, quantityL,salePriceL,
           extendedPriceL, taxL,totalL, dummyL, orderIDL;
 private JTextField
           customerName,
           transactionDesc, orderDateAndTime,
           itemDescreption, quantity,salePrice,
           extendedPrice, tax, total,orderID,
           status;
 private JButton            addToCart, deleteFromCart, submit;
 private JComboBox          customerID, transactionCode, itemID;
 private JScrollPane        jsp;
 private GridBagLayout      gbLayout;
 private GridBagConstraints gbConstraints;
 private Iots.IotsMed       myIotsMed;
 private float  wextendedPrice, taxf, totalf,wsalePrice, allTotal;
 private int    quantityint;
 private Vector headersVec, detailsVec, rowVec;
```

```
public PlaceOrderBean(Iots.IotsMed inMed) {
 myIotsMed= inMed;
 gbLayout = new GridBagLayout();
 setLayout(gbLayout);
 gbConstraints = new GridBagConstraints();
 userID      ="User1";
 headersVec = new Vector();
 headersVec.addElement("Item ID");
 headersVec.addElement("Desc");
 headersVec.addElement("Qty");
 headersVec.addElement("U_Price");
 headersVec.addElement("Ex_Price");
 headersVec.addElement("Tax");
 headersVec.addElement("Total($)");
 allTotal = 0.0f;
 String str ="_____" +
        "_____";
 dummyL        = new JLabel (str);
 dummyL.setForeground((java.awt.Color.magenta).darker());
 customerIDL    = new JLabel ("Customer ID");
 customerID     = new JComboBox();
 transactionCodeL = new JLabel ("Transaction");
 transactionCode  = new JComboBox();
 dateL          = new JLabel("Date&Time");
 orderDateAndTime = new JTextField("",12);
 orderDateAndTime.setEditable(false);
 customerName    = new JTextField("",20);
 customerName.setEditable(false);
 itemDescreption  = new JTextField("",10);
 itemDescreption.setEditable(false);
 transactionDesc  = new JTextField("",20);
 transactionDesc.setEditable(false);
 itemIDL        = new JLabel ("Item ID");
 itemDescreptionL = new JLabel ("Desc ");
 quantityL      = new JLabel ("Qty ");
 salePriceL     = new JLabel ("U_Price ");
 extendedPriceL  = new JLabel ("Ex_price ");
 taxL           = new JLabel ("Tax ");
 totalL         = new JLabel ("Total Due$");
 orderIDL       = new JLabel ("Order ID :");

 itemID         = new JComboBox();
 quantity       = new JTextField ("1",5);
```

```
    salePrice       = new JTextField ("",8);
    salePrice.setEditable(false);
    extendedPrice   = new JTextField ("",8);
    extendedPrice.setEditable(false);
    tax             = new JTextField ("",8);
    tax.setEditable(false);
    total           = new JTextField ("",8);
    total.setEditable(false);
    total.setForeground(java.awt.Color.red);
    orderID         = new JTextField ("",8);
    orderID.setEditable(false);
    orderID.setForeground(java.awt.Color.red);
    addToCart       = new JButton("Add To Cart");
    addToCart.setForeground(java.awt.Color.red);
    deleteFromCart  = new JButton("Delete From Cart");
    deleteFromCart.setForeground(java.awt.Color.red);
    submit          = new JButton("Submit Order");
    submit.setForeground(java.awt.Color.red);
    status          = new JTextField("",20);
    status.setEditable(false);
//-----------------------------------------------------------------
    detailsVec = new Vector();
//-----------------------------------------------------------------
    try {
     transactionDesc.setText("");
     transIDs = myIotsMed.getTransIDs();
     orderDateAndTime.setText(myIotsMed.getDate());
       for (int i=0; i< transIDs.length; i++){
            transactionCode.addItem(transIDs[i].sTransactionCode);
       }//for
     }
     catch (Exception ex) {
      transactionDesc.setText("Error");
     }
    transactionCode.addActionListener(new ActionListener()
        {
         public void actionPerformed(ActionEvent event) {
          try {
              transactionDesc.setText(
           transIDs[transactionCode.getSelectedIndex()].sTransactionDesc);
           }
          catch (Exception ex) {
           transactionDesc.setText("Not able to refresh");
```

```
        }
      }
      } );
//-------------------------------------------------------------------
  try {
   customerName.setText("");
   cusIDs = myIotsMed.getCusIDs();
     for (int i=0; i< cusIDs.length; i++){
         customerID.addItem(cusIDs[i].sCustomerID);
     }//for
  }
  catch (Exception ex) {
   customerName.setText("Error");
  }
  customerID.addActionListener(new ActionListener()
      {
       public void actionPerformed(ActionEvent event) {
        try {
            customerName.setText(
         cusIDs[customerID.getSelectedIndex()].sCustomerName);
        }
        catch (Exception ex) {
         customerName.setText("Not able to refresh");
        }
      }
      } );
//-------------------------------------------------------------------
  try {
   itemDescreption.setText("");
   itemIDs2 = myIotsMed.getItemIDs2();
     for (int i=0; i< itemIDs2.length; i++){
         itemID.addItem(itemIDs2[i].sItemID);
     }//for
  }
  catch (Exception ex) {
   itemDescreption.setText("Error");
  }
  itemID.addActionListener(new ActionListener()
      {
       public void actionPerformed(ActionEvent event) {
        try {
            itemDescreption.setText(
          itemIDs2[itemID.getSelectedIndex()].sItemDescreption);
```

```
        wsalePrice =
            itemIDs2[itemID.getSelectedIndex()].sSalePrice;
        quantityint = java.lang.Integer.parseInt(quantity.getText());
        salePrice.setText(
         java.lang.String.valueOf(wsalePrice));

        wextendedPrice =
         (itemIDs2[itemID.getSelectedIndex()].sSalePrice) *
          (quantityint);
        extendedPrice.setText(java.lang.String.valueOf(wextendedPrice));
        taxf = wextendedPrice * 0.07f;
        tax.setText(java.lang.String.valueOf(taxf));
        totalf = wextendedPrice + taxf;
        total.setText(java.lang.String.valueOf(allTotal));
        }
       catch (Exception ex) {
        itemDescreption.setText("Not able to refresh Names");
       }
     }
     } );
quantity.addActionListener(new ActionListener()
     {
      public void actionPerformed(ActionEvent event) {
       try {
           wsalePrice =
         itemIDs2[itemID.getSelectedIndex()].sSalePrice;
        salePrice.setText(
         java.lang.String.valueOf(wsalePrice));
        quantityint =(java.lang.Integer.parseInt(quantity.getText()));
        wextendedPrice =
         (itemIDs2[itemID.getSelectedIndex()].sSalePrice) *
         (quantityint);
        extendedPrice.setText(java.lang.String.valueOf(wextendedPrice));
         taxf = wextendedPrice * 0.07f;
        tax.setText(java.lang.String.valueOf(taxf));
       totalf = wextendedPrice + taxf;
        total.setText(java.lang.String.valueOf(allTotal));
        }
       catch (Exception ex) {
        itemDescreption.setText("not able to do Quantity chang effect");
        }
     }
     } );
```

```
//-------------------------------------------------------------------
  addToCart.addActionListener(new ActionListener()
       {
        public void actionPerformed(ActionEvent event) {
         try {
       status.setText("Adding  to cart..");
        rowVec     = new Vector();
        rowVec.addElement(
         java.lang.String.valueOf(itemID.getSelectedItem()));
        rowVec.addElement(itemDescreption.getText());
        rowVec.addElement(new Integer(quantityint));
        rowVec.addElement(new Float(wsalePrice));
        rowVec.addElement(new Float(wextendedPrice));
        rowVec.addElement(new Float(taxf));
        rowVec.addElement(new Float(totalf));
        allTotal = allTotal + totalf;
        total.setText(java.lang.String.valueOf(allTotal));
        detailsVec.addElement(rowVec);
       JTable jt = new JTable(detailsVec,headersVec);
       jsp      = new JScrollPane(jt);
       gbConstraints.fill = GridBagConstraints.BOTH;
       addComponent(jsp    ,7,0,7,3);
       gbConstraints.fill = GridBagConstraints.NONE;
        updateUI();
        }
           catch (Exception ex) {
            status.setText("Exception Add to cart/" +ex);
           }
            }
       } );
//-------------------------------------------------------------------
  deleteFromCart.addActionListener(new ActionListener()
       {
       public void actionPerformed(ActionEvent event) {
         try {
                  status.setText("Delete From cart..");
       float wtot = 0.0f;
       for ( int i= 0; i< (detailsVec.size());i++) {
          Vector rowVec = (Vector) detailsVec.elementAt(i);
          if (java.lang.String.valueOf(rowVec.elementAt(0)) ==
             java.lang.String.valueOf(itemID.getSelectedItem()))
          {
           wtot = java.lang.Float.valueOf(
```

```
            java.lang.String.valueOf(rowVec.elementAt(6))).floatValue();
            detailsVec.removeElementAt(i);
            allTotal = allTotal - wtot;
            break;
           }
         }//for detailsVec
         if (detailsVec.isEmpty()) { allTotal =0;}
         total.setText(java.lang.String.valueOf(allTotal));
        JTable jt = new JTable(detailsVec,headersVec);
        jsp      = new JScrollPane(jt);
        gbConstraints.fill = GridBagConstraints.BOTH;
        addComponent(jsp    ,7,0,7,3);
        gbConstraints.fill = GridBagConstraints.NONE;
        updateUI();
          }
          catch (Exception ex) {
            status.setText("Exception in delete from cart" +ex);
          }
        }
        } );
//---------------------------------------------------------------------
  submit.addActionListener(new ActionListener()
        {
         public void actionPerformed(ActionEvent event) {
         try {
         Iots.OrderStruct[] inOrder =
                new Iots.OrderStruct[detailsVec.size()];
         float taxAll = 0.0f;
         for ( int i= 0; i< (detailsVec.size());i++) {
          Vector rowVec = (Vector) detailsVec.elementAt(i);
          taxAll = taxAll +
            java.lang.Float.valueOf(
             java.lang.String.valueOf(rowVec.elementAt(5))).floatValue();
          inOrder[i] = new Iots.OrderStruct(
            java.lang.String.valueOf(rowVec.elementAt(0)),
            java.lang.String.valueOf(rowVec.elementAt(1)),
            java.lang.Integer.parseInt(
              java.lang.String.valueOf(rowVec.elementAt(2))),
            java.lang.Float.valueOf(
             java.lang.String.valueOf(rowVec.elementAt(3))).floatValue(),
            java.lang.Float.valueOf(
             java.lang.String.valueOf(rowVec.elementAt(4))).floatValue(),
            java.lang.Float.valueOf(
```

```
            java.lang.String.valueOf(rowVec.elementAt(5))).floatValue(),
           java.lang.Float.valueOf(
            java.lang.String.valueOf(rowVec.elementAt(6))).floatValue()
           );
        }//for detailsVec
        stat = new Iots.StatStruct(
             userID,
             java.lang.String.valueOf(
             transactionCode.getSelectedItem()),
             java.lang.String.valueOf(
             customerID.getSelectedItem()),
             orderDateAndTime.getText(),
             taxAll,
             allTotal
             );
        ordRes = myIotsMed.placeOrder( inOrder, stat);
        orderID.setText(ordRes.sOrderID);
        status.setText(ordRes.sMessage);
        }
          catch (Exception ex) {
           status.setText("Sorry: Can't Submit incomplete order" +ex);
          }
      }
      } );
//---------------------------------------------------------------------
  gbConstraints.fill = GridBagConstraints.BOTH;
  gbConstraints.anchor=GridBagConstraints.NORTHWEST;
  addComponent(customerIDL    ,0,0,1,1);
  addComponent(customerID     ,0,1,1,1);
  addComponent(transactionCodeL,0,2,1,1);
  addComponent(transactionCode ,0,3,1,1);
  addComponent(dateL          ,0,4,1,1);
  addComponent(orderDateAndTime,0,5,1,1);
  addComponent(customerName   ,1,0,3,1);
  addComponent(transactionDesc ,1,3,4,1);
  addComponent(itemIDL        ,2,0,1,1);
  addComponent(itemDescreptionL,2,1,1,1);
  addComponent(quantityL      ,2,2,1,1);
  addComponent(salePriceL     ,2,3,1,1);
  addComponent(extendedPriceL  ,2,4,1,1);
  addComponent(taxL           ,2,5,1,1);
  addComponent(itemID         ,3,0,1,1);
  addComponent(itemDescreption ,3,1,1,1);
```

```
  addComponent(quantity       ,3,2,1,1);
  addComponent(salePrice      ,3,3,1,1);
  addComponent(extendedPrice  ,3,4,1,1);
  addComponent(tax            ,3,5,1,1);
  gbConstraints.fill = GridBagConstraints.CENTER;
  addComponent(dummyL         ,4,0,7,1);
  gbConstraints.fill = GridBagConstraints.BOTH;
  addComponent(totalL         ,5,0,1,1);
  addComponent(total          ,5,1,1,1);
  addComponent(addToCart      ,5,2,1,1);
  addComponent(deleteFromCart ,5,3,2,1);
  addComponent(submit         ,5,5,2,1);
  addComponent(orderIDL       ,6,0,1,1);
  addComponent(orderID        ,6,1,1,1);
  addComponent(status         ,6,2,5,1);
  setVisible(true);
 }// constructor
  private void addComponent( Component c, int row, int column, int width,
                    int height) {
  gbConstraints.gridx = column;
  gbConstraints.gridy = row;
  gbConstraints.gridwidth= width;
  gbConstraints.gridheight= height;
  gbLayout.setConstraints(c, gbConstraints);
  add(c);
 }//addComponent
}//PlaceOrderBean
```

L. TRACKORDERBEAN

```
//TrackOrderBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class TrackOrderBean extends JPanel {
 private JPanel mainPl, detailsPl, statusPl,lDatePl,hDatePl;
// private Iots.TransactionStruct[] transIDs;
 private JLabel     orderIDL, lDateL,hDateL, instL1,instL2, orderIDcL;
```

```
private JTextField orderID,   status;
private JButton          trackOrder, getOrdrDetails,getStatusDetails;
private JComboBox        orderIDc,lDateDD,lDateMMM,lDateYY,
                  hDateDD,hDateMMM,hDateYY;
private JScrollPane      jsp;
private String [] months = {"JAN","FEB","MAR","APR","MAY","JUN",
                   "JUL","AUG","SEP","OCT","NOV","DEC"};
private GridBagLayout       gbLayout;
private GridBagConstraints   gbConstraints;
private Iots.IotsMed        myIotsMed;
private Vector              mainVec, detailsVec, statusVec,
                    mainHVec,detailsHVec,statusHVec;

public TrackOrderBean(Iots.IotsMed inMed) {
 myIotsMed= inMed;
 gbLayout = new GridBagLayout();
 setLayout(gbLayout);
 gbConstraints = new GridBagConstraints();
 mainHVec = new Vector();
 mainHVec.addElement("Order ID");
 mainHVec.addElement("Customer ID");
 mainHVec.addElement("Order Date");
 mainHVec.addElement("Invoice ID");
 mainHVec.addElement("Tax");
 mainHVec.addElement("Total Due($)");
 mainHVec.addElement("Status");
 detailsHVec = new Vector();
 detailsHVec.addElement("Item ID");
 detailsHVec.addElement("Desc");
 detailsHVec.addElement("Qty");
 detailsHVec.addElement("Unit Price");
 detailsHVec.addElement("Extended Price");
 statusHVec = new Vector();
 statusHVec.addElement("Status Date");
 statusHVec.addElement("Status");
 statusHVec.addElement("Remarks");
 orderIDL       = new JLabel("Order ID");
 orderID        = new JTextField("",8);;
 lDateL         = new JLabel("Low Date");
 hDateL         = new JLabel("High Date");
 lDatePl        = new JPanel();
 hDatePl        = new JPanel();
 lDateDD        = new JComboBox();
```

```
    lDateMMM        = new JComboBox();
    lDateYY         = new JComboBox();
    hDateDD         = new JComboBox();
    hDateMMM        = new JComboBox();
    hDateYY         = new JComboBox();
    for (int i=1; i<32; i++) {
     lDateDD.addItem(java.lang.String.valueOf(i));
     hDateDD.addItem(java.lang.String.valueOf(i));
    }
    for (int i=0; i< 20; i++) {
     lDateYY.addItem(java.lang.String.valueOf(i));
     hDateYY.addItem(java.lang.String.valueOf(i));
    }
    for (int i=0; i< months.length; i++) {
     lDateMMM.addItem(months[i]);
     hDateMMM.addItem(months[i]);
    }
    hDateDD.setSelectedIndex(30);
    hDateMMM.setSelectedIndex(11);
    hDateYY.setSelectedIndex(19);
    lDatePl.add(lDateDD);
    lDatePl.add(lDateMMM);
    lDatePl.add(lDateYY);
    hDatePl.add(hDateDD);
    hDatePl.add(hDateMMM);
    hDatePl.add(hDateYY);
    trackOrder      = new JButton("Track Order");
    trackOrder.setForeground((java.awt.Color.red).darker());
    String ins1     = "To track a single order, enter the OrderID " +
                      "then press Track Order Button.";
    instL1          = new JLabel (ins1);
    String ins2     = "Enter Low&High date ranges, clear OrderID, and " +
                      "press Track Order to track orders within that date range.";
    instL2          = new JLabel (ins2);
    status          = new JTextField("",20);
    status.setEditable(false);
    orderIDc        = new JComboBox();
    orderIDc.addItem(" ");
//-----------------------------------------------------------------
    gbConstraints.fill = GridBagConstraints.BOTH;
    gbConstraints.anchor=GridBagConstraints.NORTHWEST;
    addComponent(orderIDL  ,0,0,1,1);
    addComponent(orderID   ,0,1,1,1);
```

```
  addComponent(lDateL    ,0,2,1,1);
  addComponent(lDatePl   ,0,3,1,1);
  addComponent(hDateL    ,0,4,1,1);
  addComponent(hDatePl   ,0,5,1,1);
  addComponent(trackOrder,1,0,2,1);
  addComponent(instL1    ,1,2,4,1);
  addComponent(instL2    ,2,0,7,1);
  //3,0 - 6,0 reserved mainPl
  //7,0      reserved orderIDc,orderIDcL
  //8,0 -10,0 reserved statusPl

  addComponent(status    ,11,0,7,1);
  setVisible(true);
//---------------------------------------------------------------------
  trackOrder.addActionListener(new ActionListener()
       {
       public void actionPerformed(ActionEvent event) {
        try {
                status.setText("Tracking Order..");
        mainPl     = new JPanel();
        orderIDcL  = new JLabel("Select Order ID for more details");
        Iots.OrderMainInfoStruct [] orders;
        String lDate = java.lang.String.valueOf
                (lDateDD.getSelectedItem()) +"-"+
                java.lang.String.valueOf
                (lDateMMM.getSelectedItem()) +"-"+
                java.lang.String.valueOf
                (lDateYY.getSelectedItem());
        String hDate = java.lang.String.valueOf
                (hDateDD.getSelectedItem()) +"-"+
                java.lang.String.valueOf
                (hDateMMM.getSelectedItem()) +"-"+
                java.lang.String.valueOf
                (hDateYY.getSelectedItem());
        orders = myIotsMed.getOrderMainInfo(
           orderID.getText(), lDate, hDate);
        mainVec = new Vector();
        Vector  rowVec;
        orderIDc.removeAllItems();
        for ( int i=0; i<orders.length; i++) {
         orderIDc.addItem(orders[i].sOrderID);
         rowVec = new Vector();
         rowVec.addElement(orders[i].sOrderID);
```

```
            rowVec.addElement(orders[i].sCustomerID);
            rowVec.addElement(orders[i].sOrderDateAndTime);
            rowVec.addElement(orders[i].sInvoiceID);
            rowVec.addElement(orders[i].sTax);
            rowVec.addElement(orders[i].sTotalDue);
            rowVec.addElement(orders[i].sStatus);
            mainVec.addElement(rowVec);
           }//for
          JTable jt = new JTable(mainVec,mainHVec);
          jt.setRowHeight(13);
          jt.setPreferredScrollableViewportSize( new Dimension(400,30));
          jsp = new JScrollPane(jt);
          //mainPl.add(jsp);
          addComponent(jsp      ,3,0,7,3);
          addComponent(orderIDc ,7,0,1,1);
          addComponent(orderIDcL,7,2,5,1);
          updateUI();
                }
             catch (Exception ex) {
               status.setText("Exception in getting states// " +ex);
             }
              }
         } );
//---------------------------------------------------------------------
   orderIDc.addActionListener(new ActionListener()
         {
          public void actionPerformed(ActionEvent event) {
           try {
              status.setText("Tracking Order..");
           statusPl     = new JPanel();
           Iots.StatusStruct[] states;
           states = myIotsMed.getStatus(
             java.lang.String.valueOf(orderIDc.getSelectedItem()) );
           statusVec = new Vector();
           Vector  rowVec;
           for ( int i=0; i<states.length; i++) {
            rowVec = new Vector();
            rowVec.addElement(states[i].sStatusDate);
            rowVec.addElement(states[i].sStatus);
            rowVec.addElement(states[i].sRemarks);
            statusVec.addElement(rowVec);
           }//for
          JTable jt = new JTable(statusVec,statusHVec);
```

```
        jt.setRowHeight(13);
        jt.setPreferredScrollableViewportSize( new Dimension(400,100));
        jsp = new JScrollPane(jt);
        addComponent(jsp     ,8,0,7,3);
        updateUI();
              }
           catch (Exception ex) {
             status.setText("Exception in Tracking order" +ex);
           }
            }
       } );
//-----------------------------------------------------------------------
 }// constructor
 private void addComponent( Component c, int row, int column, int width,
                    int height) {
  gbConstraints.gridx = column;
  gbConstraints.gridy = row;
  gbConstraints.gridwidth= width;
  gbConstraints.gridheight= height;
  gbLayout.setConstraints(c, gbConstraints);
  add(c);
 }//addComponent
}//TrackOrderBean
```

L. BALANCEBEAN

```
//BalanceBean.java
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.String;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
public class BalanceBean extends JPanel {
 private JPanel pl;
 private JLabel itemIDL, partNumberL, taxableItemL,salePriceL,
           unitOfSaleL, warehouseIDL, balanceL,
           reorderLevelL, reorderQtyL, separatorL;
 private JTextField
           itemDescreption, partNumber, taxableItem,
```

```
        salePrice, unitOfSale, warehouseName,
        balance, reorderLevel, reorderQty, status;
private JButton          getBalanceb;
private JComboBox        warehouseID, itemID;
private GridBagLayout    gbLayout;
private GridBagConstraints  gbConstraints;
private Iots.balanceStruct  balStruct;
private Iots.IotsMed     myIotsMed;
public BalanceBean(Iots.IotsMed inMed) {
 myIotsMed= inMed;
 gbLayout = new GridBagLayout();
 setLayout(gbLayout);
 gbConstraints = new GridBagConstraints();
 itemIDL     = new JLabel ("Item ID");
 partNumberL  = new JLabel ("Part No");
 taxableItemL = new JLabel ("Taxable?");
 salePriceL   = new JLabel ("Sales Price $");
 separatorL   = new JLabel("_____________");
 unitOfSaleL  = new JLabel ("Unit of Sale");
 warehouseIDL = new JLabel ("Warehouse ID");
 balanceL     = new JLabel ("Balance");
 reorderLevelL= new JLabel ("Reorder level");
 reorderQtyL  = new JLabel ("Reorder Qty");
 itemDescreption= new JTextField("",25);
 itemDescreption.setEditable(false);
 partNumber    = new JTextField("",15);
 partNumber.setEditable(false);
 taxableItem   = new JTextField("",1);
 taxableItem.setEditable(false);
 salePrice     = new JTextField("",12);
 salePrice.setEditable(false);
 unitOfSale    = new JTextField("",2);
 unitOfSale.setEditable(false);
 warehouseName  = new JTextField("",25);
 warehouseName.setEditable(false);
 balance       = new JTextField("",12);
 balance.setEditable(false);
 balance.setForeground(java.awt.Color.red);
 reorderLevel   = new JTextField("",12);
 reorderLevel.setEditable(false);
 reorderQty    = new JTextField("",12);
 reorderQty.setEditable(false);
 status        = new JTextField("",50);
```

```
warehouseID    = new JComboBox();
warehouseID.addItem("    ");
try {
 Iots.IDsStruct[] wareIDs;
 wareIDs = myIotsMed.getWareIDs();
   for (int i=0; i< wareIDs.length; i++){
        warehouseID.addItem(wareIDs[i].sWareID);
   }//for
 }
 catch (Exception ex) {
 status.setText("Could not construct the wareIDs comboBox/ " +ex);
 }
itemID    = new JComboBox();
itemID.addItem("    ");
try {
 Iots.ItemIDsStruct[] itemIDs;
 itemIDs = myIotsMed.getItemIDs();
   for (int i=0; i< itemIDs.length; i++){
        itemID.addItem(itemIDs[i].sItemID);
   }//for
 }
 catch (Exception ex) {
 status.setText("Could not construct the itemIDs comboBox/ " +ex);
 }
getBalanceb    = new JButton("Get Balance");
getBalanceb.setForeground(java.awt.Color.red);
getBalanceb.addActionListener(new ActionListener()
     {
      public void actionPerformed(ActionEvent event) {
       try {
              status.setText("");
          balStruct = myIotsMed.getBalance
              (java.lang.String.valueOf(itemID.getSelectedItem()),
               java.lang.String.valueOf(warehouseID.getSelectedItem()));
          warehouseID.setSelectedItem(balStruct.sWarehouseID);
          warehouseName.setText(balStruct.sWarehouseName);
          itemID.setSelectedItem(balStruct.sItemID);
          itemDescreption.setText(balStruct.sItemDescreption);
          partNumber.setText(balStruct.sPartNumber);
          taxableItem.setText(balStruct.sTaxableItem);
          salePrice.setText(
              java.lang.String.valueOf(balStruct.sSalePrice));
          unitOfSale.setText(balStruct.sUnitOfSale);
```

```
            balance.setText(
                java.lang.String.valueOf(balStruct.sBalance));
            reorderLevel.setText(
                java.lang.String.valueOf(balStruct.sReorderLevel));
            reorderQty.setText(
                java.lang.String.valueOf(balStruct.sReorderQty));
              }
            catch (Exception ex) {
             status.setText("Exception getBalance: " +ex.toString());
            }
             }
        } );
 gbConstraints.fill = GridBagConstraints.NONE;
 gbConstraints.anchor=GridBagConstraints.NORTHWEST;
 addComponent(itemIDL,0,0,1,1);
 addComponent(itemID, 0,1,1,1);
 addComponent(warehouseIDL,0,2,1,1);
 addComponent(warehouseID,0,3,1,1);
 addComponent(itemDescreption,1,0,2,1);
 addComponent(warehouseName,1,2,2,1);
 addComponent(partNumberL ,2,0,1,1);
 addComponent(partNumber  ,2,1,1,1);
 addComponent(taxableItemL,2,2,1,1);
 addComponent(taxableItem ,2,3,1,1);
 addComponent(salePriceL ,3,0,1,1);
 addComponent(salePrice  ,3,1,1,1);
 addComponent(unitOfSaleL,3,2,1,1);
 addComponent(unitOfSale ,3,3,1,1);
 addComponent(separatorL, 4,0,4,1);
 addComponent(balanceL,5,0,1,1);
 addComponent(balance ,5,1,1,1);
 gbConstraints.fill = GridBagConstraints.BOTH;
 addComponent(getBalanceb,5,2,2,3);
 gbConstraints.fill = GridBagConstraints.NONE;
 addComponent(reorderLevelL,6,0,1,1);
 addComponent(reorderLevel ,6,1,1,1);
 addComponent(reorderQtyL  ,7,0,1,1);
 addComponent(reorderQty   ,7,1,1,1);
 addComponent(status,8,0,4,1);
 setVisible(true);
}// constructor
private void addComponent( Component c, int row, int column, int width,
                    int height) {
```

```
    gbConstraints.gridx = column;
    gbConstraints.gridy = row;
    gbConstraints.gridwidth= width;
    gbConstraints.gridheight= height;
    gbLayout.setConstraints(c, gbConstraints);
    add(c);
  }//addComponent
}//BalanceBean
```

LIST OF REFERENCES

Akbay, M., Lewis, S., *Design and Implementation of an Enterprise Information System Utililzing a Component Based Three-tier Client/Server Database System*, 1999.

Blaha, M., Premerlani, W., *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, 1998.

Booch, G., *Object-Oriented Analysis and Design with Applications*, Second Edition, Addison-Wesley, 1994.

Deitel, H., Deitel, P., *Java How to Program*, Second Edition, Prentice Hall, 1998.

Emasri, R., Navathe, S., *Fundamentals of Database Systems*, Second Edition, Addison-Wesley, 1994.

Hamilton, G., Cattell, R., Fisher, M., *JDBC Database Acess with Java*, Addison-Wesely, 1997.

Henning, M., Vinoski, S., *Advanced CORBA Programming with C++*, Addison-Wesely, 1999.

Hoffer, J., George, J., Valacich, S., *Modern System Analysis and Design*, Benjamin/ Cummings Publishing Company, Inc., 1996.

Inprise, *Gatekeeper Guide*, Inprise Corporation, 1999.

Kroenke, D., *Database Processing Fundementals, Design, and Implementation*, Sixth Edition, Prentice Hall, 1999.

Lewis,G., Barber, S., Siegel, E., Programming with Java IDL, John Wiley and Sons, Inc., 1998.

Norman, W., *What is XML*, 1998. www.xml.com/pup/98/10/guide.html

Nissen, M.E., *Agent-Based Supply Chain Integration*, Journal of Information Technology Management Special Issue on E-Commerce in Procurement and the Supply Chain (forthcoming 2000).

Orfali, R., Harkey, D., *Client/Server Programming with Java and CORBA*, Second Edition, John Wiley and Sons, Inc., 1998.

Slama, Dirk, Garbis, Jason, Russsell, Perry, *Enterprise CORBA*, Prentice Hall, 1999.

Taylor, A., *JDBC Developer's Resource,* Second Edition, Prentice Hall, 1999.

Treese, G., WinField, Lawrence, C., Stewart, *Designing Systems for Internet Commerce*, Addison-Wesley, 1998.

Umar, Amjad, *Application (Re)engineering Building Web-based Applications and Dealing with Legacies*, Prentice Hall, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Prof. Daniel Dolk (Code IS/DK) 1
   Naval Postgraduate School
   Monterey, CA 93943

4. Professor James Bret Michael (Code CS/MB) 1
   Naval Postgraduate School
   Monterey, CA 93943

5. COL Saleem Abu Diyah 1
   Jordanian Air Force
   North Marka - Amman
   Jordan

6. Ahmed Ali Falah Almandeel Otoom 1
   Alborge - Soof
   Jerash
   Jordan

7. Computer Center / Jordanian Air Force 2
   North Marka - Amman
   Jordan

8. Chairman, Code CS 1
   Naval Postgraduate School
   Monterey, CA 93943